TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JANI NEVALAINEN

RAKENNUKSEN 3D-TIETOMALLIPALVELIN

Diplomityö

Tarkastaja: professori Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa 4. helmikuuta 2015

TIIVISTELMÄ

NEVALAINEN, JANI: RAKENNUKSEN 3D-TIETOMALLIPALVELIN
Tampereen teknillinen yliopisto
Diplomityö, 86 sivua, 13 liitesivua
Tammikuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Tommi Mikkonen

Avainsanat: IFC, tietomalli, tietomallipalvelin

Rakennushankkeessa on monia eri osapuolia, joiden välillä on siirrettävä tietoa rakennushankkeen aikana. Aiemmin rakennushankkeissa rakenteita on kuvattu 2D-piirustusten avulla ja osapuolten välillä siirrettävä tieto on ollut yksittäisiä dokumentteja. Tiedonsiirtoon on käytetty projektipankkeja, joihin dokumentteja on tallennettu.

Nykyään suunnittelua tehdään yhä enemmän luomalla rakennettavasta kohteesta kolmiulotteisia tietomalleja. Tietomalleihin lisätään rakenteiden geometrian lisäksi paljon metatietoa. Tiedonsiirtoon käytetyt projektipankit eivät ymmärrä tietomalleihin tallennettua geometriaa tai metatietoa. Tämän takia hankkeen osapuolet eivät saa hyödynnettyä tietomallia tehokkaasti.

Tässä diplomityössä toteutetaan tietomallipalvelin, jonka avulla tiedonsiirtoa ja sen hyödyntämistä voidaan tehostaa. Tietomallipalvelin ymmärtää tietomallien sisältämää geometriaa- ja metatietoa ja tarjoaa rajapinnan, jonka avulla rakennushankkeen osapuolet pääsevät käsiksi haluamaansa tietoon. Tietomallien tallennusformaattina käytetään avoimen IFC-standardin mukaisia tietomalleja. Tietomallipalvelin toteutetaan web-sovelluksena, jotta se olisi helposti käytettävissä ilman erillisten sovellusten asentamista. Tietomallipalvelin tarjoaa lisäksi 3D-näkymän, jonka kautta tietomallin geometriaa ja metatietoja voi tarkastella.

Työn tuloksena saatiin aikaan tietomallipalvelin, joka toteuttaa työlle asetetut tavoitteet. Tietomallipalvelimelle voi tallentaa IFC-tietomalleja ja tietomallipalvelin tarjoaa toiminnot, jonka kautta käyttäjät voivat hakea ja näyttää tietomallin sisältämiä geometria- ja metatietoja.

Tietomallipalvelimen käyttökokeilussa tietomallipalvelinta hyödynnettiin osana mobiilisovellusta IFC-tietomallin metadatan säilömisessä. Käyttökokeilussa toteutettu mobiilisovellus voitti AEC Hackaton-tapahtumassa parhaan projektin palkinnon, joka todisti, että kehitetty sovellus on hyödyllinen rakennusalalle. Käyttökokeilu osoitti, että tietomallipalvelin on tarvittaessa integroitavissa muihin järjestelmiin. Tämä luo paljon uusia mahdollisuuksia ja käyttötapauksia, joissa tietomallipalvelinta voidaan hyödyntää tulevaisuudessa.

ABSTRACT

NEVALAINEN, JANI: Building information model server
Tampere University of Technology
Master of Science Thesis, 86 pages, 13 Appendix pages
January 2016
Master's Degree Programme in Information Technology
Major: Software Engineering
Examiner: Professor Tommi Mikkonen

Keywords: IFC, Building Information Model, Building Information Model server

There are many different participants in construction projects who need to exchange information with each other. Before different buildings and structures have been designed and presented using 2D-drawings and the information being transferred has been single documents. Documents have been transferred and stored using project banks.

Currently design is done more by creating building information models which are three dimensional representations of the buildings. These building information models include the building's geometry and metadata. Project banks used for information exchange do not understand the data stored in the building information models. For this reason the information stored in the building information models cannot be used efficiently.

The topic of this master's thesis is to implement building information model server which gives possibility to utilize the information inside the building information models. Server offers functionalities which enable different participants to get access to the geometry and metadata information they require. Building information models are being stored on the server using the open standard called IFC. Server is being implemented as a web service, so it can be used easily without any additional software installations. Server offers 3D-view which can be used to view the geometry and the metadata of the building information model.

As a result of this master's thesis a building information model server was implemented. It meets the requirements that were set in the beginning. It offers possibility to store the building information models based on the IFC standard on the server. It also offers possibility to search and display the geometry and metadata inside the building information model.

As a test experiment the building information model server was being used as a part of a mobile application. In the experiment the server was used to store the metadata of building information model. The mobile application won first prize in AEC Hackaton which proved that the application is useful for construction industry. The test experiment also proved that building information model server can be integrated to other systems as well. This creates more possibilities and use cases where building information model server can be used in the future.

ALKUSANAT

Haluan kiittää työpaikkaani A-Insinöörit Suunnittelu Oy:tä diplomityön aiheesta ja diplomityön ohjaajaa Tommi Mikkosta asiantuntevasta ohjauksesta. Erityisesti haluan kiittää perhettäni ja ystäviäni tuesta ja kannustuksesta.

Tampereella, 8.12.2015

Jani Nevalainen

SISÄLLYSLUETTELO

1. JOHDANTO

Rakennushankkeeseen liittyy monia eri osapuolia, joiden välillä siirretään tietoa rakennushankkeen aikana. Aiemmin suunnittelua on tehty 2D-piirustusten avulla ja tällöin osapuolten välillä on siirretty lähinnä dokumentteja. Tiedonsiirron apuna on käytetty projektipankkeja, joihin suunnitteluun liittyviä dokumentteja on tallennettu.

Nykyään yhä useammissa rakennushankkeissa rakennettavasta kohteesta luodaan kolmiulotteinen tietomalli. Tietomalli sisältää rakennuksen geometrian lisäksi myös paljon metatietoa. Metatiedon avulla voidaan antaa lisätietoja rakennettavasta kohteesta tai sen avulla voidaan kuvata yksittäisen rakenneosan ominaisuuksia, kuten sen materiaalia tai massaa.

Koko rakennuksen tietomalli koostuu eri suunnittelualojen tuottamista osamalleista, joista luodaan lopulta koko rakennusta kuvaava yhdistelmämalli. Esimerkiksi rakennesuunnittelija tuottama tietomalli pitää sisällään vain rakennuksen rakenteet, mutta ei talotekniikkaa.

Tämä asettaa uusia vaatimuksia ja haasteita myös osapuolten väliselle tiedonsiirrolle. Aiemmin tiedonsiirtoon käytetyt projektipankit eivät ymmärrä tietomalleihin tallennettua metatietoa tai geometriaa, jonka takia tietomalleja ei saada hyödynnettyä tehokkaasti. Tämän takia tarvitaan uusia järjestelmiä, jotka ymmärtävät siirrettävää tietoa paremmin ja tarjoavat myös rajapinnan sen hyödyntämiseen.

Tiedonsiirtoa ja käsittelyä voidaan tehostaa tietomallipalvelimen avulla. Tietomallipalvelimella tarkoitetaan tietokantapohjaista järjestelmää, johon voidaan tallentaa rakennuksen tietomalleja. Tietomallit tallennetaan tietokantaan niin, että esimerkiksi yksittäiset rakenneosat ovat tarvittaessa erotettavissa. Tietomallipalvelin voidaan toteuttaa asiakas-palvelin arkkitehtuuria hyödyntäen. Tällöin tietomallipalvelin voi tarjota rajapinnan, joka on käytettävissä monilta eri päätelaitteilta. Rajapinnan avulla eri rakennushankkeen osapuolet pääsevät hakemaan ja päivittämään tietomallin tietoja. Tietomallipalvelin voi tarjota tarkastelun avuksi visuaalisen näkymän, jossa näkyy tietomallin sisältämä geometria. Tietomallipalvelin voi myös auttaa voi tietomallin tiedonhallinnassa ja tarjota toiminnot erilaisten rakennushankkeessa tarvittavien raporttien luomiseen.

On olemassa useita tietomallipalvelin toteutuksia, joiden avulla rakennusten tietomalleja voidaan tarkastella ja jakaa. Useimmat markkinoilla olevat tietomallipalvelimet keskitty-

vät arkkitehtisuunnitteluun. Koska A-Insinöörit Suunnittelu Oy:n projektit ovat suurimmalta osin rakennesuunnittelua, yrityksessä nähtiin tarve omalle toteutukselle, jossa on otettu huomioon rakennesuunnittelijan näkökulma.

Tässä diplomityössä toteutetaan tietomallipalvelin, jonka tarkoituksena on vastata A-Insinöörit Suunnittelu Oy:n tarpeisiin. Toinen tärkeä asia, joka loi tarpeen tietomallipalvelimen kehitykselle, on yrityksessä suunniteltujen tietomallien varmuuskopioinnin parantaminen. Tietomallipalvelimen avulla varmuuskopioinnin palautusta voidaan tehostaa ja nopeuttaa.

Rakennuksen tietomalli ja tietomallipalvelin käsitteet kuvataan tarkemmin luvussa kaksi. Luvussa esitellään myös tietomallien tallennukseen tarkoitettu avoin IFC-standardi. Lopuksi luvussa esitellään olemassa olevien tietomallipalvelimien toteutuksia.

Luvussa kolme käydään läpi tietomallipalvelimelle asetetut vaatimukset asiakkaan näkökulmasta. Luvussa esitetään syyt, jotka ovat luoneet tarpeen tietomallipalvelimen kehittämiselle. Luvussa esitellään myös A-Insinöörit Suunnittelu Oy:n tarpeiden pohjalta luodut toiminnalliset vaatimukset tietomallipalvelimelle sekä järjestelmälle asetetut laadulliset vaatimukset ja rajoitteet. Luvussa tarkastellaan myös tietomallipalvelimeen liittyvät järjestelmät.

Luvussa neljä esitellään tietomallipalvelimen toteutustekniikat. Luvussa esitellään yleisellä tasolla mihin valittuja toteutustekniikoita käytetään järjestelmän toteutuksessa sekä käydään läpi toteutuksessa hyödynnettävät sovelluskehykset. Luvussa esitellään sovelluskehystä laajentavat kolmannen osapuolen kirjastot, käytettävä ohjelmointikieli ja kehitysympäristö. Luvussa esitellään myös käyttöliittymän toteutukseen käytettäviä tekniikoita ja taustalla käytettävä tietokanta.

Luvussa viisi esitellään kuinka valittuja toteutustekniikoita on käytetty tietomallipalvelimen toteutuksessa. Luvussa käydään läpi järjestelmän arkkitehtuuri ja tietomallipalvelimen toiminnallisuudet. Luvussa esitellään myös tietomallipalvelimen taustalla olevan tietokannan tietorakenne sekä käyttäjähallinnan toteutus. Lopuksi käydään läpi kuinka ulkoinen liityntä Tekla Structures-suunnitteluohjelmistoon on toteutettu.

Luvussa kuusi esitellään toteutetun tietomallipalvelimen käyttökokeilu. Luvussa tarkastellaan myös valituissa toteutustekniikoissa havaittuja hyötyjä ja ongelmia. Lopuksi luvussa käydään läpi järjestelmän jatkokehitysajatuksia.

Luvussa seitsemän esitetään yhteenveto siitä mitä diplomityössä on toteutettu. Tämän lisäksi esitellään tietomallipalvelimen käyttökokeilu sekä johtopäätökset.

2. RAKENNUKSEN TIETOMALLI JA TIETOMALLI-PALVELIN

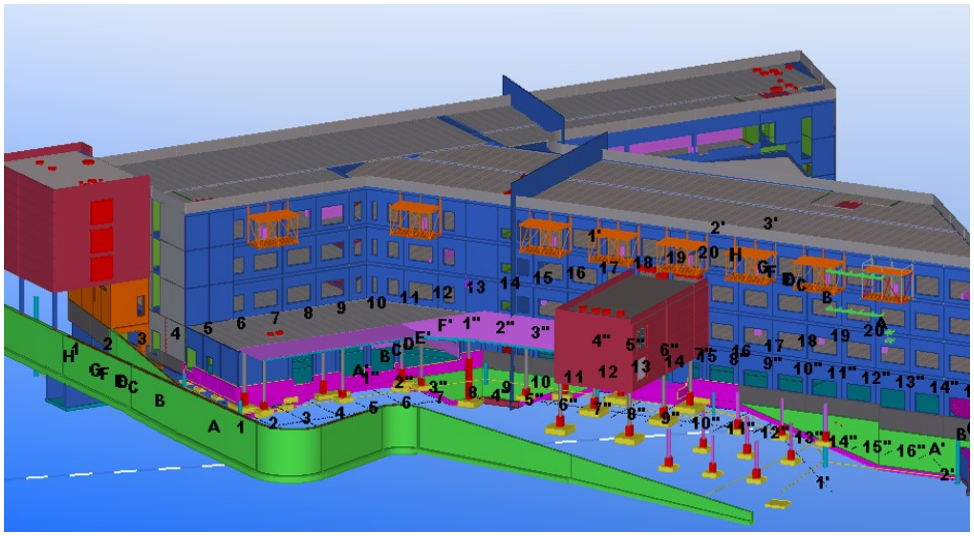
Tässä luvussa tarkastellaan mitä rakennuksen tietomallilla tarkoitetaan ja mitä tietoja se sisältää. Ensimmäisenä kuvataan mitä rakennuksen tietomalli tarkoittaa yleisellä tasolla. Seuraavaksi esitellään tietomallien tallennukseen tarkoitettu avoin IFC-standardi (IFC Overview summary). Tämän jälkeen käydään läpi mitä tietomallipalvelin käsite tarkoittaa. Lopuksi esitellään olemassa olevien tietomallipalvelimien toteutuksia ja niiden tarjoamia toimintoja.

2.1 Rakennuksen tietomalli yleisesti

Rakennuksen tietomallilla tarkoitetaan rakennuksen ja sen elinkaaren aikaisten tietojen kokonaisuutta digitaalisessa muodossa. Yleisemmin käytössä oleva englanninkielinen termi on Building Information Model, josta käytetään lyhennettä BIM (Eastman et al. 2011, Ch. 1).

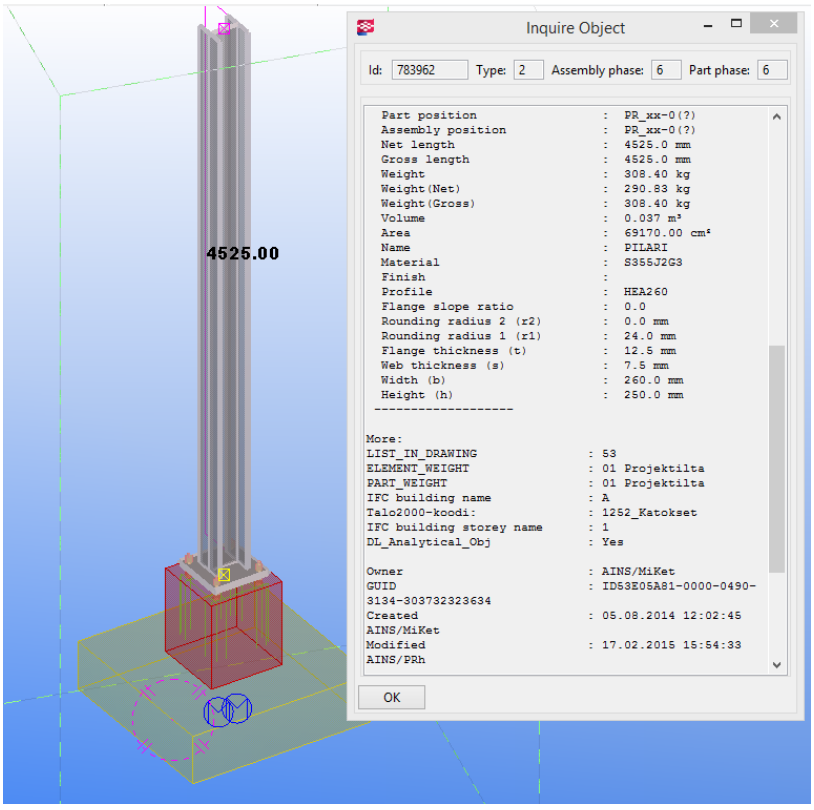
Tietomalli sisältää rakennuksen 3D-geometrian, jonka avulla voidaan muodostaa kolmiulotteinen malli rakennuksesta. Pelkkää geometriaa sisältävää kolmiulotteinen malli ei kuitenkaan ole vielä varsinainen tietomalli, sillä se kattaa vain pienen osan siitä tiedosta, mitä tarvitaan rakennussuunnittelussa. Tämän takia tietomalliin tallennetaan myös metatietoa. Metatiedon avulla voidaan antaa lisätietoja rakennettavasta kohteesta tai kuvata yksittäisten rakenneosien ominaisuuksia. Näitä ominaisuuksia voivat olla esimerkiksi rakennuksen tontin sijainti tai tietyn rakenneosan tyyppi, tuotetunniste, materiaali, massa, palo- ja rasitusluokka sekä käyttöikä. Tietomallissa rakenneosien välille voidaan luoda suhteita, jolloin saadaan muodostettua useammista osista koostuvia kokoonpanoja. Suhteiden avulla saadaan lisäksi luotua älykkäämpiä rakenneosia. Tällöin esimerkiksi, jos seinään kuuluvan osan massa muuttuu, vaikuttaa se myös koko seinän massaan. (Eastman et al. 2011, Ch. 1)

Koko rakennuksen tietomalli koostuu eri suunnittelualojen tuottamista osamalleista, joista voidaan koostaa yhdistelmämalleja. Yksittäiseen osamalliin tallennetaan vain tarvittava tieto, ja se riippuu mallin tuottajasta ja sen käyttötarkoituksesta. Esimerkiksi rakennesuunnittelijan tuottama malli pitää sisällään vain rakennuksen rakenteet, mutta ei talotekniikkaa. Seuraavalla sivulla olevassa kuvassa (Kuva 2.1) on näkyvillä Tekla Structures-suunnitteluohjelmistolla tuotettu rakennesuunnittelijan tietomalli, johon on mallinnettu rakennuksen perustukset, teräsrakenteet sekä kantavat ja ei-kantavat betonirakenteet. Tietomalli sisältää myös rakenteista luodut 2D-piirustukset. Kyseisen tietomallin koko on noin 450 megatavua.



Kuva 2.1 Tekla Structures-suunnitteluohjelmistolla tuotettu rakennesuunnittelijan tietomalli.

Alla olevassa kuvassa (Kuva 2.2) on esitetty saman tietomallin yksittäisen pilarin sisältämiä metatietoja. Pilarin metatiedot sisältävät esimerkiksi pilarin materiaalin, massan ja tilavuuden sekä tiedon pilarin luoneesta henkilöstä.



Kuva 2.2 Tekla Structures-suunnitteluohjelmistolla tuotetun rakennesuunnittelijan tietomallin yksittäisen pilarin metatietoja.

Tietomallin sisältämät tiedot päivittyvät ja täydentyvät koko rakennushankkeen ajan, ja niitä hyödynnetään eri tavoin riippuen rakennushankkeen vaiheesta. Ennen rakennushankkeen alkamista tietomallin avulla voidaan visualisoida vaihtoehtoisia toteutusvaihtoehtoja ja arvioida tulevia rakennuskustannuksia. Tietomallia voidaan käyttää myös erilaisiin simulointeihin, joilla tarkastellaan esimerkiksi rakenteiden kestävyyttä tai energiatehokkuutta. Rakennushankkeen aikana tietomallia voidaan käyttää asennusten suunnitteluun, aikataulutukseen ja hankkeen etenemisen seurantaan. Rakennuksen valmistuttua voidaan luoda toteutunutta rakennusta vastaava ylläpitomalli, jota on mahdollista käyttää hyödyksi rakennuksen ylläpidossa.

2.2 IFC

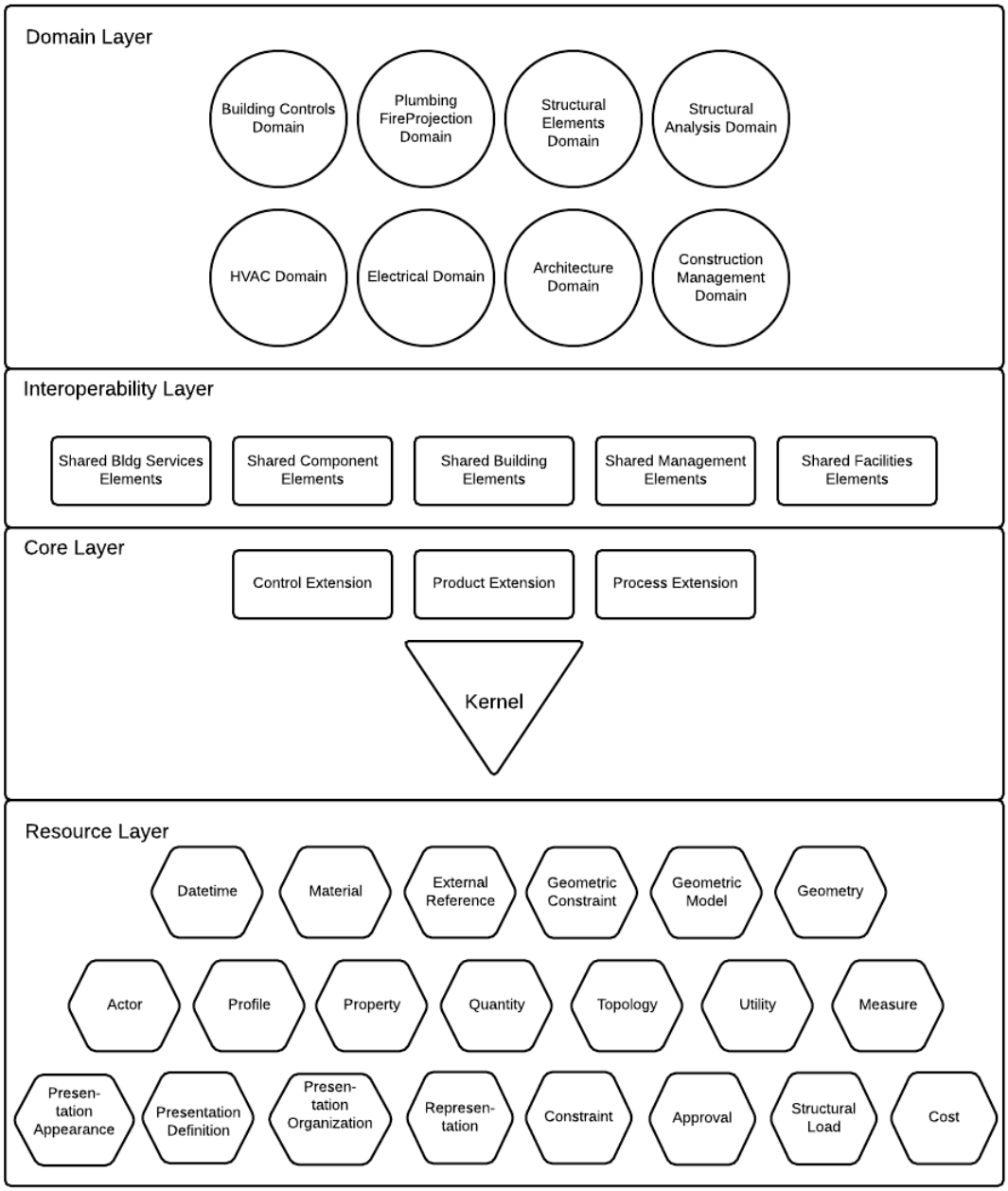
IFC eli Industry Foundation Classes on avoin standardi rakennuksen tietomallien tallentamiseen (IFC Overview summary). Standardin kehittämisestä on vastuussa BuildingSMART-organisaatio, joka tunnettiin aiemmin nimellä International Alliance for Interoperability (Eastman et al. 2011, Ch. 3). IFC-standardi sisältää määrittelyjä monelle eri rakennusteollisuuden toimialueelle, mutta tässä kohdassa käydään läpi vain sen perusrakenne ja olennaisimmat osat tämän työn kannalta.

Standardia lähdettiin kehittämään, koska rakennusteollisuudessa oli tarvetta avoimelle tiedostomuodolle, jota voitaisiin hyödyntää koko rakennuksen elinkaaren ajan. Standardin avulla haluttiin myös parantaa tiedonsiirtoa eri suunnitteluohjelmistojen välillä. Standardin kehittäminen aloitettiin vuonna 1994, ja siitä on olemassa useita eri versioita, joista uusin julkaistu versio on IFC4. (Eastman et al. 2011, Ch. 3; IFC Overview summary) Tällä hetkellä kaikki suunnitteluohjelmistot eivät kuitenkaan vielä tue uusinta versiota, joten yleisemmin käytössä oleva versio on IFC2x3 TC1.

IFC:n määrittely perustuu ISO 10303- ja ISO 10303-21-standardeihin, jotka on tarkoitettu valmistavan teollisuuden tuotemallien kuvaamiseen ohjelmistoriippumattomasti. Standardit kattavat useita eri teollisuudenaloja, ja niiden tavoitteena on helpottaa tiedonsiirtoa eri toimijoiden välillä. ISO 10303-21-standardista käytetään yleisemmin nimeä STEP, joka tulee englanninkielen sanoista Standard for the Exchange of Product Model Data. IFC-standardi on kuvattu EXPRESS-määrittelykielen avulla, joka on määritetty ISO 10303-11-standardissa. (Eastman et al. 2011, Ch. 3)

IFC:n arkkitehtuuri käyttää hyväkseen periyttämistä. Tämä tarkoittaa sitä, että päätasolla määritellyistä geneerisemmistä tietorakenteista voidaan periyttää uusia tarkemmin määriteltyjä tietorakenteita tarpeen mukaan.

IFC:n järjestelmäarkkitehtuuri ja siitä löytyvät osiot ovat näkyvillä seuraavalla sivulla olevassa kuvassa (Kuva 2.3). Arkkitehtuurin kannalta keskeisimmät määrittelyt sijaitsevat Resource- ja Core Layer-tasoilla.



Kuva 2.3 IFC4-standardin järjestelmäarkkitehtuurin kuvaus (perustuu lähteeseen IFC4 Specification).

Alimmalla tasolla sijaitseva Resource Layer sisältää uudelleenkäytettävät perusresurssit, joilla voidaan esittää mm. geometriaa, joukkoja ja objektien tiettyjä ominaisuuksia. Perusresursseja voidaan käyttää hyödyksi muodostettaessa ylemmän tason objekteja. (Eastman et al. 2011, Ch. 3; IFC4 Specification)

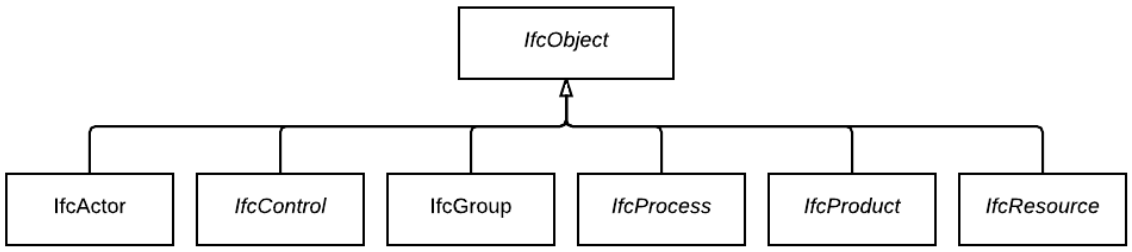
Seuraavalla tasolla oleva Core Layer on yksi arkkitehtuurin keskeisimmistä osista. Kernel-osiossa määritetään abstrakti kantaluokka `IfcRoot`, josta ylemmän tason peruskäsitteet kuten objekti (`IfcObject`), ominaisuus (`IfcPropertyDefinition`) ja suhde (`IfcRelationship`) on periytetty. IFC-standardi kuvaa rakennuksen ja sen rakenneosat objekteina, joilla on ominaisuuksia ja suhteita.

IfcRoot-luokka sisältää myös attribuutit GlobalId, Name ja Description, jotka löytyvät kaikilta Object-luokasta periytetyiltä olioilta. GlobalId-attribuutin avulla kaikki IFC-standardissa määriteltyjen luokkien ilmentymät voidaan yksikäsittäisesti identifioida. Name ja Description-attribuutteja käytetään ilmentymien nimeämiseen ja kuvaamiseen. Tasolta löytyy myös Product Extension-osio, jossa määritellään abstraktit kantaluokat, joista esimerkiksi varsinaiset rakenneosat periytetään. (Eastman et al. 2011, Ch. 3; IFC4 Specification)

Toiseksi ylimmällä tasolla oleva Interoperability Layer sisältää niiden rakenneosien määrittelyt, joita hyödynnetään monilla eri rakennussuunnittelun osa-alueilla. Tällaisia rakenneosia ovat esimerkiksi Shared Building Elements-osiossa määritetyt seinä, pilari, palkki ja ovi. Toinen tärkeä tällä tasolla oleva osio on Shared Facilities Management, joka sisältää määrittelyt huonekaluille. (IFC4 Specification)

Ylimmällä tasolla on Domain Layer, joka sisältää rakennussuunnitteluun liittyvien toimialojen omat määrittelyt. Määrittelyt sisältävät kyseiselle suunnittelualueelle tarvittavia laajennettuja resursseja. Tasolta löytyvät omat määrittelyt esimerkiksi arkkitehdeille, rakennesuunnitteluun ja talotekniikkaan. (IFC4 Specification)

Objektien määritysten abstraktina kantaluokkana toimivasta IfcObject-luokasta periytetyt luokat ovat näkyvissä alla olevassa kuvassa (Kuva 2.4).



Kuva 2.4 IfcObject-luokan perintähierarkia (perustuu lähteeseen IFC4 Specification).

IfcObject-luokasta on edelleen periytetty kuusi eri luokkaa, joita käytetään eri tyyppisten objektien kuvaamiseen. Useimmat näistä luokista ovat abstrakteja kantaluokkia, joista varsinaiset luokkien ilmentymät on periytetty.

IfcActor-luokkaa ja siitä periytettyjä luokkia käytetään rakennusprojektiin liittyvien ihmisten ja organisaatioiden kuvaamiseen. Määrittely sisältää esimerkiksi nimet, osoitteet ja roolit. (IFC4 Specification, Ch. 5)

IfcControl-luokan määrittelyjen avulla voidaan kuvata rakennussuunnitteluun liittyviä rajoitteita. Tällaisia rajoitteita voivat olla esimerkiksi rakennesuunnittelussa käytetyt suunnittelustandardit. (IFC4 Specification, Ch. 5)

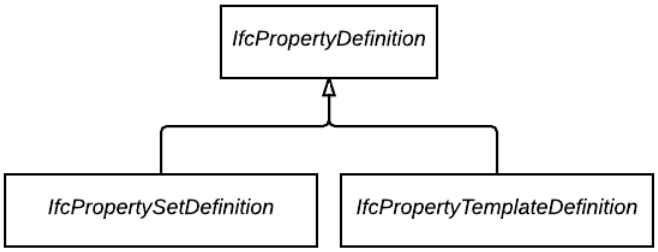
IfcGroup-luokkaa käytetään hyväksi joukkojen kuvaamiseen. Joukkoja voidaan muodostaa kaikista IfcObject-luokasta periytetyistä luokista. Tämä mahdollistaa myös sisäkkäisten joukkojen kuvaamisen. (IFC4 Specification, Ch. 5)

IfcProcess on abstrakti kantaluokka, jonka määrittelyjen avulla kuvataan rakennusvaiheeseen tai rakennuksen ylläpitoon liittyviä prosesseja. Prosessimäärittelyt käyttävät hyväkseen suhteita, joiden avulla voidaan määritellä esimerkiksi mihin rakenneosiin prosessi liittyy ja mitä resursseja siihen on käytössä. (IFC4 Specification, Ch. 5)

IfcProduct-luokka on abstrakti kantaluokka niille luokille, joiden avulla voidaan kuvata rakennuksessa olevat fyysiset objektit, joilla on geometria. Tästä luokasta on periytetty esimerkiksi rakenneosien määrittelyt. Luokasta on periytetty myös ei-fyysisten objektien määrittelyyn käytettyjä luokkia, joilla voidaan kuvata esimerkiksi rakennusprojektia, rakennustyömaata, rakennuksen tiloja ja sen sisältämiä kerroksia. (IFC4 Specification, Ch. 5)

IfcResource on abstrakti kantaluokka luokille, joilla kuvataan rakennusprojektin käytössä olevia resursseja, kuten työvoima ja laitteistot. Luokan avulla käytössä olevia resursseja ei tarvitse välttämättä mallintaa fyysisinä objekteina tietomalliin, mutta tieto niistä kuitenkin voidaan esittää. (IFC4 Specification, Ch. 5)

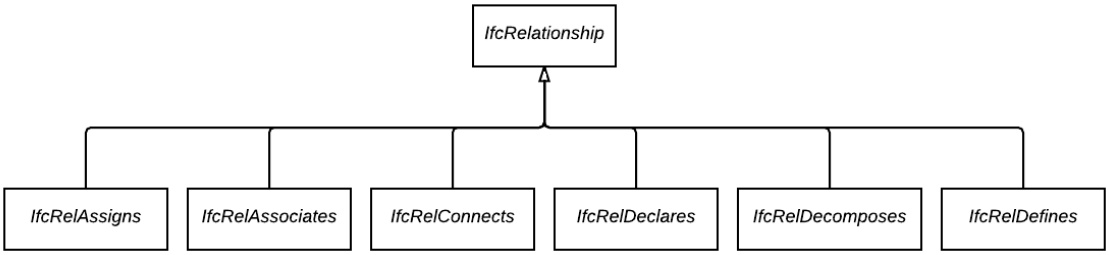
Objektien ominaisuuksia kuvataan abstraktista kantaluokasta IfcPropertyDefinition periytettyjen luokkien avulla. Luokasta on periytetty kaksi luokkaa IfcPropertySetDefinition ja IfcPropertyTemplateDefinition. IfcPropertyDefinition luokan perintähierarkia on näkyvillä alla olevassa kuvassa (Kuva 2.5). (IFC4 Specification, Ch. 5)



Kuva 2.5 IfcPropertyDefinition-luokan perintähierarkia (perustuu lähteeseen IFC4 Specification).

IfcPropertySetDefinition-luokan kuvaamat ominaisuudet voivat olla dynaamisesti määriteltyjä tai staattisia. Dynaamisesti määritellyt ominaisuudet antavat mahdollisuuden määritellä objektien ominaisuuksia tietomalliin vapaasti käyttäen avain-arvo- pareja. Staattiset ominaisuudet ovat IFC-standardissa ennalta määriteltyjä. IfcPropertyTemplateDefinition-luokan avulla voidaan muodostaa ominaisuusjoukkoja, joita voidaan hyödyntää projektin objekteille yhteisesti. (IFC4 Specification, Ch. 5)

Objektit ja niiden ominaisuudet yhdistetään IFC-standardissa suhteiden avulla. Objektien ja ominaisuuksien välisiä suhteita kuvaavat luokat on periytetty abstraktista kantaluokasta `IfcRelationship`. Yhteyssuhteet voivat olla yhden suhde yhteen tai yhden suhde moneen. (IFC4 Specification, Ch. 5). Kantaluokasta on periytetty edelleen kuusi eri abstraktia kantaluokkaa, joita voidaan käyttää erityyppisten suhteiden kuvaamiseen. Näissä periytetyissä kantaluokissa on määritetty attribuutteja, kuten `RelatedObjects`, joihin toisiinsa liittyvät objektit voi asettaa. `IfcRelationship`-luokan perintä hierarkia on esitetty alla olevassa kuvassa (Kuva 2.6).



Kuva 2.6 IfcRelationship-luokan perintähierarkia (perustuu lähteeseen IFC4 Specification).

`IfcRelAssings`-luokasta periytettyjen luokkien avulla voidaan määrittää prosessien ja niihin liittyvien resurssien yhteyssuhteita. Yhteyssuhteen avulla voidaan myös määrittää mihin rakenneosaan resurssi liittyy.

`IfcRelAssociates`-luokasta periytettyjä luokkia voidaan käyttää suhteiden luontiin, jolla objekteille voidaan linkittää ulkoista informaatiota. Linkitettävä tieto voi olla esimerkiksi rakennesuunnitteluun liittyviä suunnittelu- tai tarkastusdokumentteja.

`IfcRelConnects`-luokka toimii kantaluokkana luokille, joilla voidaan asettaa rajoitteita objektien välisille yhteyssuhteille. Määritysten avulla voidaan luoda säännöstö siitä mitkä objektit voivat muodostaa yhteyssuhteita keskenään.

`IfcRelDeclares`-luokasta periytettyjen luokkien avulla määritellään, missä kohtaa tietomallin määrittelyä tietyt objektit voivat esiintyä. (IFC4 Specification, Ch. 5) Näin muodostetaan myös IFC-tietomallin hierarkkinen rakenne. Tietomallissa päätasolla on aina projekti (`IfcProject`), jonka alla on rakennustyömaa (`IfcSite`), joka sisältää rakennuksen (`IfcBuilding`). Rakennus pitää sisällään kerroksia (`IfcBuildingStorey`), jotka sisältävät tiedon kyseisessä kerroksessa olevista rakennuselementeistä. (Eastman et al. 2011, Ch. 3; IFC4 Specification Ch. 5)

`IfcRelDecomposes` on kantaluokka luokille, joiden avulla voidaan muodostaa yhteyssuhteita objektien välille. Tällä tavoin rakennuselementit sisältävät tiedon muista niihin liittyvistä elementeistä ja niistä voidaan muodostaa esimerkiksi kokoonpanoja. (Eastman et al. 2011, Ch. 3; IFC4 Specification Ch. 5)

`IfcRelDefines`-luokasta periytettyjä luokkia käytetään yhdistämään samoja ominaisuuksia sisältäviä objekteja. Esimerkiksi useat rakennuksen ikkunat voivat olla ominaisuuksiltaan samanlaisia. Tällöin niiden ominaisuudet tarvitsee määritellä vain kerran ja ominaisuudet voidaan linkittää yhteyssuhteen avulla. (IFC4 Specification, Ch. 5)

IFC-tiedosto voidaan tallentaa kolmessa eri formaatissa. Ensimmäinen ja yleisemmin käytössä oleva formaatti on .ifc, joka käyttää tallentamiseen STEP-rakennetta. Toinen formaatti on .ifcXML, jossa tietomalli tallennetaan XML-dokumenttina (Extensible Markup Language). XML-muotoa käytetään yleensä vähemmän, koska sen avulla luodut tiedostot ovat kooltaan huomattavasti suurempia muihin tallennusmuotoihin verrattuna. Kolmas formaatti on .ifcZIP, jossa .ifc- tai .ifcXML-tiedosto pakataan PKzip 2.04g-algoritmin avulla. Tämän formaatin avulla saavutetaan huomattavasti pienempi tiedostokoko kuin muilla tavoilla. (IFC Overview Summary)

Alla olevassa kuvassa (Kuva 2.7) on näkyvissä esimerkki IFC-standardin mukaisesta STEP-muodossa tallennetusta tiedostosta, jolla kuvataan yksittäinen palkki. Tiedostosta on karsittu selvyyden vuoksi osa riveistä.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition[CoordinationView_V2.0]'),'2;1');
FILE_NAME('single_beam.ifc','2015-01-26T12:02:27',('AINS\\JaNev'),('Structural Designer'),'IFC Database Version:Work...
FILE_SCHEMA(('IFC2X3'));
ENDSEC;

DATA;
#1= IFCPERSON('AINS\\JaNev','Undefined',$,$,$,$,$,$);
#2= IFCORGANIZATION($,'Tekla Corporation',$,$,$);
#3= IFCPERSONANDORGANIZATION(#1,#2,$);
#4= IFCAPPLICATION(#2,'20.0 Service Release 5','Tekla Structures Educational','Multi material modeling');
#5= IFCOWNERHISTORY(#3,#4,$,.NOCHANGE.,$,$,$,1422266547);
...
#14= IFCSIUNIT(*,.LENGTHUNIT.,.MILLI.,.METRE.);
...
#24= IFCPROJECT('1b1ewnBgL4axQq6aeC2$7B',#5,$,$,$,$,$,(#11),#23);
#25= IFCLOCALPLACEMENT($,#10);
#26= IFCSITE('0GMw7zEmrF_Qm08cQ_nCyg',#5,'Undefined',$,$,#25,$,$,.ELEMENT.,$,$,0.,$,$);
#27= IFCLOCALPLACEMENT(#25,#10);
#28= IFCBUILDING('2r7kG87oXCZvP_rBqKu61u',#5,'Undefined',$,$,#27,$,$,.ELEMENT.,$,$,$);
#29= IFCLOCALPLACEMENT(#27,#10);
#30= IFCBUILDINGSTOREY('3WTUo5kWHCdhMquTKIwEF_',#5,'Undefined',$,$,#29,$,$,.ELEMENT.,0.);
...
#50= IFCBEAM('1KnX1X000LkZ4qCZ8oDZOr',#5,'BEAM','IPE200','IPE200',#33,#49,'ID54c61061-0000-15ba-3134-323232363635');
#51= IFCBEAMTYPE('1YWRqQoGb8MAJsJUDr6VTZ',#5,'IPE200',$,$,$,$,$,.NOTDEFINED.);
...
#60= IFCPROPERTYSINGLEVALUE('Grade',$,IFCLABEL('S355J2'),$);
#61= IFCPROPERTYSINGLEVALUE('Profile',$,IFCLABEL('IPE200'),$);
#62= IFCPROPERTYSINGLEVALUE('Name',$,IFCLABEL('BEAM'),$);
#63= IFCPROPERTYSET('2SyhSckjj7lgoqaEl8x9kS',#5,'Tekla Common','Common Properties to Shared building elements',(#52...
...
#79= IFCMATERIAL('STEEL/S355J2');
#80= IFCRELAGGREGATES('1IRqJfRc91yeBeJFdjWXcF',#5,$,$,#24,(#26));
#81= IFCRELAGGREGATES('0SPJBrOXD43eUjhjI9_j1O',#5,$,$,#26,(#28));
#82= IFCRELAGGREGATES('2puVv_52z7uA0C38qFSAz3',#5,$,$,#28,(#30));
#83= IFCRELCONTAINEDINSPATIALSTRUCTURE('2txLkjXzHBghmlkfnG2JhC',#5,$,$,(#50),#30);
#84= IFCRELDEFINESBYPROPERTIES('0D8UrQqDz5cg8RNdlPKycg',#5,$,$,(#50),#63);
#86= IFCRELDEFINESBYTYPE('2T8I1KzZj6gub5Z9n0mqS$',#5,$,$,(#50),#51);
#87= IFCRELASSOCIATESMATERIAL('2R0gSRy513IB1j54FjYfKW',#5,$,$,(#50),#79);
#88= IFCPRESENTATIONLAYERASSIGNMENT('BEAM',$,(#47),$);
ENDSEC;

END-ISO-10303-21;
```

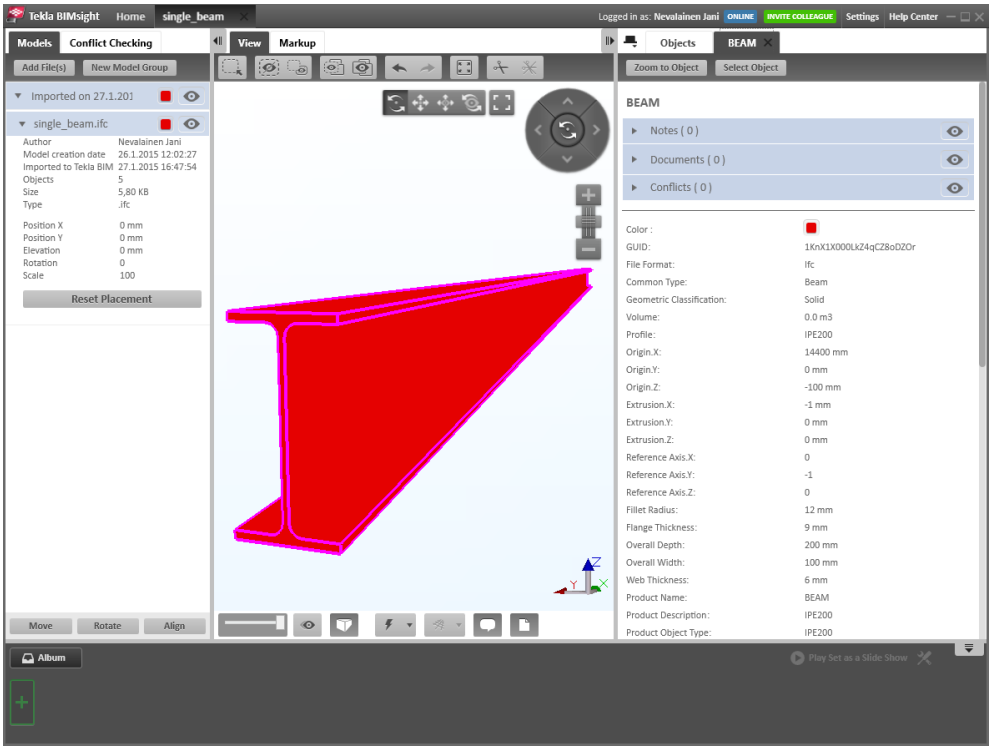
Kuva 2.7 IFC-standardin mukainen STEP-muodossa tallennettu yksittäinen palkki.

STEP-formaatti kuvaa IFC-standardin luokkien ilmentymät riveittäin. Rivin alussa on #-merkillä aloitettu tunnistenumero, jonka jälkeen esitetään luokan nimi ja sen sisältämien attribuuttien määrittelyt. Attribuutit on rajattu sulkumerkein ja erotettu toisistaan pilkulla. Attribuuttien määrä on vakio ja määrittelemättömät attribuutit voidaan esittää $-merkillä. Attribuuttien sisältämät viittaukset muihin luokkien ilmentymiin esitetään tunnistenumeron avulla.

Esimerkkitiedoston alussa on nähtävillä normaalit IFC-standardin mukaiset otsikkotiedot, joissa määritetään esimerkiksi käytetty versio. Otsikkotietojen jälkeen alkaa data-osuus. Dataosuuden alusta löytyvät IFC-standardissa määritetyt tietomallin pakolliset luokat kuten `IfcProject`, `IfcSite`, `IfcBuilding` ja `IfcBuildingStorey`. Näiden luokkien välisten suhteiden ja hierarkian kuvaamiseen on käytetty `IfcRelAggregates`-luokkia.

Palkki on määritelty `IfcBeam`-luokan avulla. Palkin ominaisuuksien määrittelyyn on käytetty `IfcPropertySingleValue`-luokkaa. Palkki ja sen ominaisuudet on yhdistetty `IfcPropertySet`-luokkien avulla. Palkin materiaali on kuvattu `IfcMaterial`-luokan avulla ja se on yhdistetty palkkiin `IfcRelAssociatesMaterial`-luokan avulla. Tieto palkin liittymisestä kerrokseen on kuvattu `IfcRelContainedInSpatialStructure`-luokan avulla.

Esimerkin kuvailema palkki Tekla Bimsight -sovelluksessa on näkyvissä alla olevassa kuvassa (Kuva 2.8). Sovelluksen oikeassa reunassa näkyvät ominaisuustiedot. Osa ominaisuuksista on näkyvissä myös esimerkkitiedostossa.



Kuva 2.8 IFC-standardin mukainen STEP-muodossa tallennettu yksittäinen palkki Tekla Bimsight-sovelluksessa.

Vaikka IFC-standardia on kehitetty pitkään, ei se vieläkään sisällä aivan kaikkea rakennushankkeissa tarvittavaa informaatiota. Kehitys kuitenkin jatkuu ja standardi laajenee edelleen. Tulevaisuudessa standardiin on tarkoitus sisällyttää esimerkiksi siltasuunnitteluun liittyviä määrityksiä (buildingSMART Current projects).

2.3 Tietomallipalvelin

Eastman et al. (2011, Ch. 3) mukaan tietomallipalvelimella tarkoitetaan tietokantapohjaista järjestelmää, johon voidaan tallentaa rakennuksen tietomalleja ja niiden sisältämää tietoa. Tietomallit tulisi tallentaa tietokantaan niin, että esimerkiksi yksittäiset rakenneosat olisi tarvittaessa erotettavissa. Eastman et al. (2011, Ch. 3) esittää, että jokaisella tietomallin objektilla, joka tallennetaan tietokantaan, tulisi olla yksikäsitteinen tunniste eli GUID, jonka avulla se voidaan tunnistaa. Tällä voidaan edesauttaa eri suunnittelualueiden tuottamien tietomallien uusien versioiden synkronointia. Tämä on yksi olennaisimmista hyödyistä, mitä tietomallipalvelimen avulla voidaan saavuttaa. (Eastman et al. 2011, Ch. 3)

Jotta tietomallien synkronointia eri suunnittelualueiden välillä voidaan suorittaa, tulee tietomallipalvelimen tarjota rajapinta, jonka avulla tietomallissa olevaa tietoa voidaan hakea ja päivittää (Eastman et al. 2011, Ch. 3). Hietasen (2005, s. 50) mukaan tietomallipalvelimen kanssa kommunikointi voidaan toteuttaa asiakaspalvelin-periaatteella, joka mahdollistaa tiedonsiirron molempiin suuntiin.

Tietomallipalvelimen tulisi tarjota myös toimintoja, joiden avulla tietomallille voidaan tehdä erilaisia automaattisia tarkistuksia ja analyysejä. Näiden toimintojen avulla voitaisiin varmistua siitä, että tietomallin sisältämä tieto on eheää ja että se vastaa sille asetettuja vaatimuksia. Tietomallipalvelimen on lisäksi mahdollistettava käyttäjähallinta, jotta voidaan hallita sitä kuka palvelimella olevaan tietoon pääsee käsiksi ja kuka sen omistaa. (Eastman et al. 2011, Ch. 3)

Tietomallien sisältämän tiedon eheys ja tietomallien synkronointi ei ole ainoa hyöty, mitä tietomallipalvelimen avulla voidaan saavuttaa verrattuna perinteiseen projektipankkiin. Hietasen (2005, s. 50) mukaan tietomallipalvelinta käytettäessä käyttäjä voi määrittää missä muodossa tiedon haluaa nähdä. Jos esimerkiksi tietomallipalvelimelta löytyy arkkitehdin suunnittelema tietomalli, voi käyttäjä halutessaan pyytää raporttia rakennuksen pinta-aloista rakennuksen kerrosten mukaan jäsennettynä. Projektipankkia käytettäessä käyttäjän täytyy erikseen pyytää arkkitehtiä tekemään raportti ja julkaisemaan se projektipankissa. (Hietanen 2005, s. 50)

Eastman et al. (2011) jakaa tietomallipalvelimet kolmeen eri luokkaan niiden tarjoamien toiminnallisuuksien perusteella. Ensimmäiset keskittyvät rakennushankkeen projektinhallintaan. Niiden tarkoituksena on helpottaa muutostenhallintaa ja tehostaa tiedonsiirtoa

rakennushankkeen eri osapuolien välillä. Toisessa luokassa ovat tuotteen elinkaaren hallintaan keskittyvät tietomallipalvelimet, joita voidaan käyttää hyväksi esimerkiksi rakennuselementtien tuotannossa. Kolmannessa luokassa ovat rakennuksen ylläpitoon keskittyvät tietomallipalvelimet. (Eastman et al. 2011, Ch. 3)

2.4 Olemassa olevat tietomallipalvelimet

On olemassa useita tietomallipalvelin toteutuksia, joiden avulla rakennusten tietomalleja voi tarkastella ja jakaa. Tietomallipalvelimien tarjoamat toiminnot ja ominaisuudet vaihtelevat suuresti. Tämä johtuu siitä, että tietomallipalvelimet ovat yleensä toteutettu jonkun tietyn suunnittelualueen näkökulmasta. Tässä kohdassa on esitelty muutamia olemassa olevia toteutuksia, jotka on toteutettu erityisesti rakennushankkeen tarpeita ajatellen.

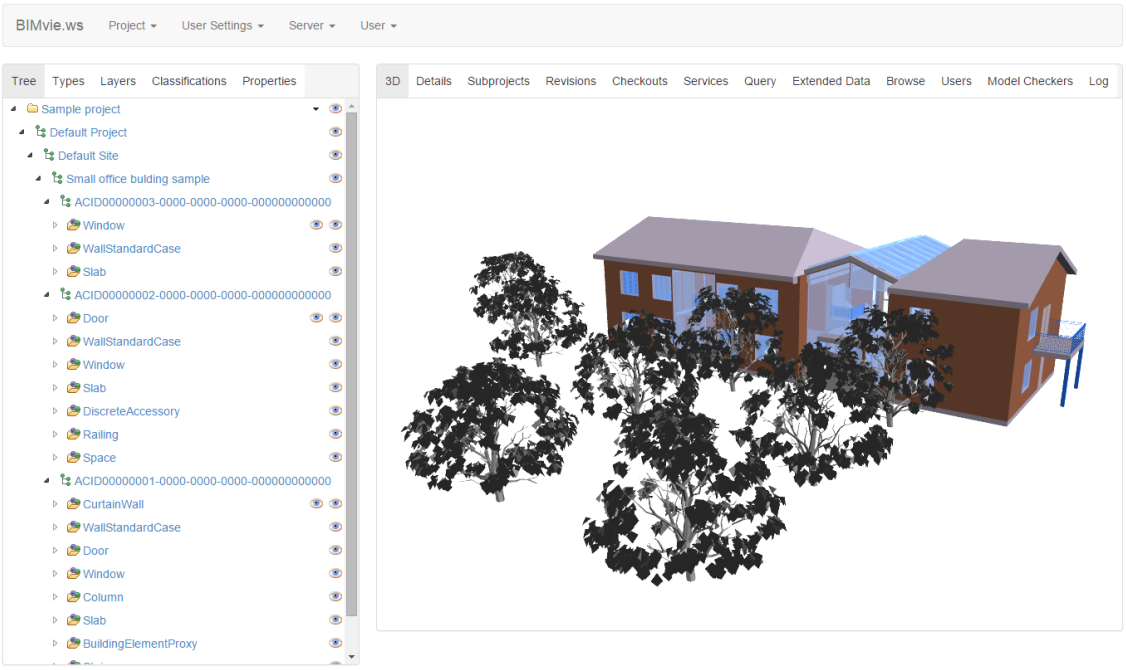
2.4.1 BIMserver

BIMserver on avoimen lähdekoodin tietomallipalvelin, jolla voi tarkastella ja käsitellä IFC-tietomalleja. Tietomallipalvelimen kehitys aloitettiin vuonna 2008. Kehittämisen aloittivat alankomaalainen TNO-organisaatio ja Eindhovenin teknillinen yliopisto (BIMserver About). Koska kyseessä on avoimen lähdekoodin projekti, nykyään toteutuksessa on mukana myös useita yksityishenkilöitä.

BIMserver on toteutettu Java-ohjelmointikielellä. Tietomallipalvelimen taustalla toimii BerkeleyDB-tietokanta. Järjestelmän arkkitehtuuri sallii myös muiden tietokantojen käytön, mutta tällä hetkellä niistä ei ole olemassa yleisesti saatavia toteutuksia. Tietomallipalvelimen toimintoja ja palvelimella olevia tietomalleja hallitaan selaimen kautta. Selaimessa toimiva hallintasovellus käyttää hyväkseen BIMServerin tarjoamaa SOAP- ja JSON-rajapintaa. Saman rajapinnan avulla myös erilliset asiakassovellukset voivat kommunikoida tietomallipalvelimen kanssa. (Beetz et al. 2010; BIMserver Wiki; JavaScript Object Notation; SOAP)

Yksi tunnetuimmista valmiista asiakassovelluksista on selaimessa toimiva BIMsurfer. BIMsurfer tarjoaa 3D-näkymän tietomallin tarkasteluun. Näkymä mahdollistaa esimerkiksi tietomallissa olevien rakenneosien tietojen tarkastelun ja rakenneosien suodattamisen. Toteutus käyttää hyväkseen WebGL-teknologiaa (WebGL), joten se ei vaadi toimiakseen erillisiä liitännäisiä. (BIMsurfer)

Tietomallipalvelimen tarjoama 3D-näkymä tietomallien tarkasteluun on näkyvillä seuraavalla sivulla olevassa kuvassa (Kuva 2.9). Näkymän vasemmassa reunassa on puurakenne, jolla kuvataan näkyvissä olevan IFC-tietomallin hierarkkinen rakenne ja sen sisältämät rakenneosat. Näkymän yläreunassa on navigaatiopalkki, josta pääsee käsiksi muihin tietomallipalvelimen tarjoamiin toimintoihin.



Kuva 2.9 BIMserver tietomallipalvelimen tarjoama 3D-näkymä tietomallien tarkasteluun.

Tietomallipalvelimen sisäinen tietorakenne perustuu IFC-standardiin. Palvelimelle lähetetyt IFC-tietomallit ja niiden sisältämät rakenneosat tallennetaan tietokantaan yksittäisinä objekteina, jotka sisältävät samat määrittelyt kuin IFC-standardi. Tietorakenteen avulla voidaan varmistaa se, että tallennettava tietomallin sisältämä tieto ei muutu tai häviä tallennuksen yhteydessä. Tietorakenne mahdollistaa edelleen tietomallien muokkauksen ja yksityiskohtaisemmat haut palvelimella. Haku-toiminnon avulla tietomallista voi hakea esimerkiksi sen sisältämät ikkunat tai tietyn yksittäisen seinän. Tietorakenteen avulla on toteutettu myös tietomallien versiointi, joka toimii yksittäisten rakenneosien tasolla. (Beetz et al. 2010; Eastman et al. 2011, Ch. 3)

Tietomallipalvelin sisältää yllä mainittujen lisäksi paljon muitakin toimintoja. Useimmat tietomallipalvelimen toiminnot on toteutettu hyödyntäen plugin-rajapintaa. Plugin-rajapinta mahdollistaa edelleen omien toimintojen toteutuksen. (BIMserver Wiki, Eastman et al. 2011, Ch. 3)

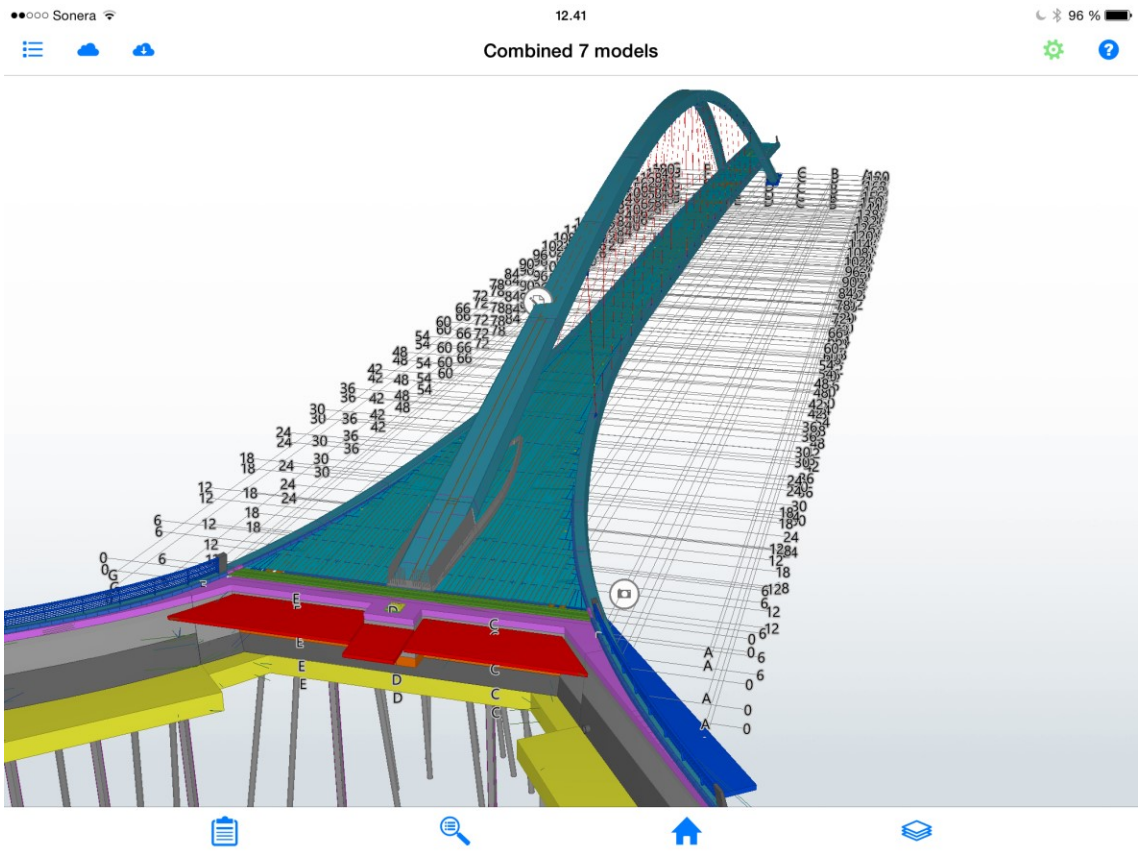
BIMserver on toiminnoiltaan yksi monipuolisimmista olemassa olevista tietomallipalvelimista. Lähdekoodia kehittävä yhteisö on melko aktiivinen ja uusia julkaisuja tulee useampi vuodessa. Lähdekoodin avoimuus mahdollistaa uusien toimintojen toteuttamisen ja sovelluksen räätälöinnin omiin tarpeisiin sopivaksi. Avoin lähdekoodi voi muodostua myös ongelmaksi kaupallisessa käytössä, kun lähdekoodia ei haluta julkistaa. Jos BIMserverin kanssa kommunikoi avoimen rajapinnan kautta, omaa toteutusta ei tosin tarvitse julkistaa (BIMserver License).

2.4.2 Tekla Field 3D

Tekla Field 3D on IFC-tietomallien tarkasteluun kehitetty sovellus. Se on alun perin kehitetty Applen iOS-käyttöjärjestelmälle ja siitä on olemassa omat versiot iPhonelle ja iPadille. Sovelluksesta löytyy myös liitännäisen avulla toimiva selainversio PC:lle. Sovelluksesta on olemassa free, premium ja enterprise versiot. Premium ja yrityksille suunnattu enterprise versio ovat maksullisia, ja ne sisältävät hieman enemmän ominaisuuksia kuin ilmainen versio. (Tekla Field 3D)

Sovelluksessa tarkasteltavat tietomallit ladataan käyttöön sähköpostista, www-osoitteesta tai yleiseltä pilvipalvelimelta. Tällä hetkellä tuki on olemassa yleisemmille pilvipalveluille kuten Dropbox (https://www.dropbox.com), Box (https://www.box.com), Gmail (https://www.gmail.com), Google Drive (https://drive.google.com) ja OneDrive (https://onedrive.live.com). Tekla Field 3D ei siis varsinaisesti ole tietomallipalvelin, vaan asiakassovellus, joka kommunikoi taustalla olevan tietomalleja sisältävän palvelimen kanssa. (Tekla Field 3D Features)

Tekla Field 3D tarjoaa 3D-näkymän tietomallien tarkasteluun. iPad-sovelluksen tarjoama näkymä tietomallien tarkasteluun on nähtävissä alla olevassa kuvassa (Kuva 2.10).



Kuva 2.10 Tekla Field 3D:n tietomallin tarkastelunäkymä.

Tietomalleja voi olla näkyvissä kerrallaan yksi tai useampi. Näkymän alareunassa on navigaatiopalkki, jonka avulla pääsee käsiksi sovelluksen tarjoamiin toimintoihin. Toiminnot mahdollistavat tietomallissa olevien rakenneosien tietojen tarkasteluun. Maksulliset versiot sisältävät enemmän toimintoja, joita ovat esimerkiksi välimatkojen mittaaminen, leikkausnäkymien luominen sekä rakenneosien haku ja suodatus. Enterprise versiossa rakenneosille on mahdollista liittää kommentteja ja kuvia, jotka näkyvät myös muille projektiin liitetyille henkilöille. (Tekla Field 3D Features)

Yrityksille tarjottava maksullinen premium-versio antaa mahdollisuuden tietomallien lähettämiseen Teklan tarjoamalle pilvipalvelimelle. Tällöin tietomallien lähetys palvelimelle tehdään selaimen kautta. Ennen lähetystä täytyy luoda projekti, jonka alle tietomallit lähetetään. Projektilla on aina yksi tai useampi pääkäyttäjä, jolla on oikeus tietomallien lähetykseen projektin alle. Peruskäyttäjät voivat ainoastaan tarkastella ja kommentoida palvelimelle lähetettyjä tietomalleja.

Enterprise-versio tarjoaa siis joitain tietomallipalvelimelta vaadittuja perustoimintoja, ja sitä voidaan hyödyntää rakennushankkeissa välittämään tietomalleja eri toimijoiden välillä. Sovellus sisältää kuitenkin monia puutteita. Haun käyttö vaatii IFC-formaatin tuntemista, sillä hakeminen tapahtuu IFC-standardissa määriteltyjen objektien nimien ja ominaisuuksien avulla. Tietomalleilla ei ole olemassa versiohistoriaa, mikä jostain muista tietomallipalvelimien toteutuksista löytyy. Sovellus ei mahdollista tarkasteltavien tietomallien säilyttämistä yrityksen sisäverkossa olevilla palvelimilla. Tämä voi muodostua ongelmaksi projekteissa, joissa suunnittelusopimukset velvoittavat säilyttämään tietomalleja vain yrityksen hallinnoimilla sisäverkossa sijaitsevilla palvelimilla.

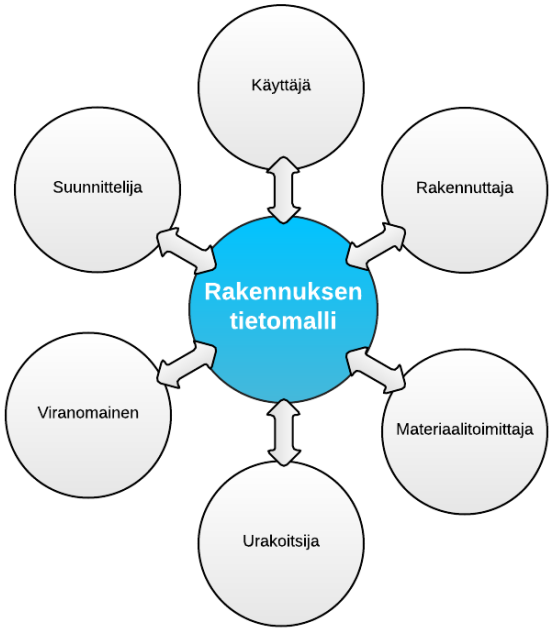
3. TIETOMALLIPALVELIMEN VAATIMUKSET

Tässä luvussa käydään läpi tietomallipalvelimelle asetetut vaatimukset asiakkaan näkökulmasta. Ensimmäisenä esitetään syyt, jotka ovat luoneet tarpeen tietomallipalvelimen kehittämiselle. Tämän jälkeen esitellään A-Insinöörit Suunnittelu Oy:n tarpeiden pohjalta luodut toiminnalliset vaatimukset tietomallipalvelimelle. Seuraavaksi esitetään järjestelmälle asetetut laadulliset vaatimukset ja rajoitteet. Lopuksi tarkastellaan tietomallipalvelimeen liittyviä järjestelmiä, jotka pitää ottaa huomioon toteutuksessa.

3.1 Tarve ja tarkoitus

Rakennushankkeeseen liittyy monia eri osapuolia, joiden välillä on siirrettävä tietoa rakennushankkeen aikana. Aiemmin rakennushankkeissa rakenteita on kuvattu 2D-piirustusten avulla. Tällöin eteenpäin siirrettävä tieto on ollut yksittäisiä dokumentteja. Näissä tapauksissa tiedonsiirtoon on käytetty esimerkiksi projektipankkeja, joihin dokumentteja on tallennettu.

Nykyään suunnittelua tehdään yhä enemmän luomalla rakennettavasta kohteesta kolmiulotteisia tietomalleja. Tietomalleihin lisätään rakenteiden geometrian lisäksi paljon metatietoa. Tämä asettaa uusia vaatimuksia ja haasteita osapuolten väliselle tiedonsiirrolle. Rakennuksen tietomallia hyödyntävät osapuolet ovat näkyvissä alla olevassa kuvassa (Kuva 3.1).



Kuva 3.1 Rakennuksen tietomallia hyödyntävät osapuolet.

Aiemmin tiedonsiirtoon käytetyt projektipankit eivät ymmärrä tietomalleihin tallennettua geometriaa tai metatietoa. Tämän takia hankkeen osapuolet eivät saa hyödynnettyä tietomallia tehokkaasti.

Toiminnan tehostamiseksi tarvitaan uusia järjestelmiä, jotka ymmärtävät siirrettävää tietoa paremmin. Tiedonsiirtoa ja sen hyödyntämistä voidaan tehostaa tietomallipalvelimen avulla. Jos tietomalli säilötään avoimessa IFC-formaatissa, tietomallipalvelinta voidaan hyödyntää tiedonsiirron lisäksi tiedonhallinnassa. Se voi tarjota toiminnot tietomallin visualisointiin ja erilaisten raporttien luomiseen.

Useimmat markkinoilla olevat tietomallipalvelimet keskittyvät arkkitehtisuunnitteluun ja arkkitehtien tarjoamien tietojen välittämiseen. Kyseiset tietomallipalvelimet eivät sisällä kaikkia rakennesuunnittelijan tarvitsemia ominaisuuksia ja toimintoja. Koska suuri osa A-Insinöörit Suunnittelu Oy:n projekteista on rakennesuunnittelua, yrityksessä nähtiin tarve omalle toteutukselle, jossa on otettu huomioon rakennesuunnittelijan näkökulma.

Toinen tärkeä asia, joka loi tarpeen tietomallipalvelimen kehitykselle, on yrityksessä suunniteltujen tietomallien varmuuskopioinnin parantaminen. Tietomallien varmuuskopiointi suoritettiin aiemmin ottamalla kopio koko binäärimuotoisesta tietomallista. Tästä johtuen varmuuskopion palautus piti suorittaa koko tietomallille kerralla, eikä sitä voinut tehdä pelkästään esimerkiksi yksittäiselle rakenneosalle. Tietomallipalvelimen avulla varmuuskopioinnin palautus voidaan tarvittaessa tehdä niin, että palautetaan vain halutut tiedot valituille rakenneosille.

3.2 Toiminnalliset vaatimukset

Tietomallipalvelimen toiminnoille asetettiin seuraavissa alakohdissa olevia vaatimuksia. Vaatimukset kehitettiin A-Insinöörit Suunnittelu Oy:n rakennesuunnitteluprojekteissa ilmenneiden ongelmien ja haasteiden pohjalta yhteistyössä yrityksessä työskentelevien rakennesuunnittelijoiden ja tietomallikoordinaattorin kanssa. Toiminnallisten vaatimusten pohjana käytettiin olemassa olevien tietomallipalvelimien sisältämiä toiminnallisuuksia, joita jalostettiin edelleen. Toiminnoista pyrittiin löytämään ne, jotka ovat olennaisia rakennesuunnitteluprojektien kannalta ja joilla voidaan tehostaa niissä tehtävää tietomallinnusta. Osalla toiminnallisuuksista on tarkoitus myös helpottaa tietomallien tiedonsiirtoa A-Insinöörit Suunnittelu Oy:n yhteistyökumppaneiden ja asiakkaiden kanssa. Suurin osa näistä vaatimuksista pyritään toteuttamaan työn edetessä. Vaatimukset on listattu prioriteettijärjestyksessä.

Vaatimus 1: Projektien perustaminen, muokkaus ja poisto. Tietomallipalvelimelle täytyy pystyä luomaan projekteja, joihin liittyy IFC-formaatissa olevia tiedostoja. Projektilla on omat ennalta määrätyt tietokentät, joita täytyy pystyä muokkaamaan myöhemmin. Projekti pitää olla myös poistettavissa, jolloin poistuvat myös siihen liittyvät IFC-tiedos-

tot. Projekteihin täytyy pystyä liittämään myös käyttäjiä. Käyttäjällä on oltava mahdollisuus listata kaikki omat projektit, joihin hänet on liitetty. Käyttäjien liittämistoimintoon liittyy käyttäjähallinta, josta on kerrottu vaatimuksessa 6.

Vaatimus 2: IFC-tiedostojen lähetys ja lataus. Tietomallipalvelimelle täytyy pystyä lähettämään IFC-formaatissa olevia tiedostoja ja niitä pitää pystyä lataamaan. Lähetetyille tiedostoille tulee tehdä eheystarkistus ja virheellisistä tiedostoista on tehtävä ilmoitus tiedoston lähettäjälle. Lähetetyille tiedostoille halutaan myöhemmin suorittaa mahdollisesti myös muita tarkastuksia, joten uusien tarkistusten lisääminen pitää ottaa huomioon tietomallipalvelimen toteutuksessa.

Vaatimus 3: Tiedostojen versiointi. Lähetetyille IFC-tiedostoille tulee toteuttaa versiointi. Käyttäjän täytyy pystyä päivittämään aiemmin lähetetty IFC-tiedosto uudemmalla versiolla. Tiedostosta pitää olla ladattavissa uusimman version lisäksi myös vanhemmat tietomallipalvelimella olevat versiot. Versioinnin avulla hoidetaan suunnitteluohjelmistolla luodun tietomallin varmuuskopiointi ja myös tietojen palautus suunnitteluohjelmistoon.

Vaatimus 4: Tietomallin geometrian visualisointi. Tietomallipalvelimen tulee ymmärtää IFC-tiedoston rakenneosien geometriaa ja sitä täytyy pystyä tarkastelemaan visuaalisesti. Visualisoinnin avulla tulee voida näyttää rakenteen lisäksi myös suunnittelun valmiusaste.

Vaatimus 5: Tietomallin metatietojen haku. Tietomallipalvelimen tulee ymmärtää IFC-tiedoston rakenneosien sisältämää metatietoa ja toteuttaa toiminto, jonka avulla metatietoja voi hakea. Haun tulokset voidaan esittää tekstimuodossa tai näyttämällä hakua vastaavat rakenneosat ja niiden geometria. Hakutoimintoon täytyy pystyä luomaan myös erilaisia ennalta määrättyjä kiinteillä hakusanoilla olevia hakuja. Ennalta määrättyjä hakuja voidaan käyttää hyväksi esimerkiksi rakenneosien määrälaskennassa, kun arvioidaan rakennuskustannuksia.

Vaatimus 6: Käyttäjähallinta. Tietomallipalvelimella oleviin projekteihin, tiedostoihin ja tietoihin ei saa päästä käsiksi ilman kirjautumista, joten tietomallipalvelimella tulee olla käyttäjähallinta. Käyttäjätasoja täytyy olla ainakin pääkäyttäjä, yrityksen työntekijä ja asiakas. Eri käyttäjien pääsyä tiedostoihin ja toimintoihin tulee voida rajata pääkäyttäjän toimesta. Esimerkiksi A-Insinöörit Suunnittelu Oy:n omilla työntekijöillä tulee olla oikeudet nähdä kaikki omat projektit, IFC-tiedostojen kaikki versiot ja niihin liittyvät tiedostot. Asiakkaat pääsevät näkemään vain rajatut projektit ja dokumentit sekä viimeisimmän version IFC-tiedostosta. Oletuksena vain pääkäyttäjät voivat luoda ja poistaa projekteja sekä tiedostoja. Pääkäyttäjän täytyy kuitenkin voida antaa projektikohtaisia oikeuksia toimintoihin muille käyttäjille. Pääkäyttäjien täytyy myös pystyä liittämään haluamiaan käyttäjiä projekteihin.

Vaatimus 7: Rajapinta. Tietomallipalvelimen tulee tarjota rajapinta ainakin osalle sen sisältämistä toiminnoista, jotta tietomallipalvelimen kanssa voi kommunikoida myös muilla sovelluksilla. Rajapinnan kautta täytyy pystyä toteuttamaan ainakin tiedostojen lähetys-, lataus-, IFC-tiedoston geometrian haku- ja metatietojen haku-toiminnot.

Vaatimus 8: Dokumentin tai URI:n linkittäminen rakenneosille. Tietomallipalvelimella olevien IFC-tiedostojen rakenneosille tulee voida linkittää yksi tai useampi dokumentti tai URI. Jos kyseessä on dokumentti, se täytyy pystyä lähettämään tietomallipalvelimelle. Tietomallipalvelimelle lähetetyt dokumentit täytyy pystyä lataamaan. Linkitettyjen dokumenttien tulisi säilyä rakenneosalla myös seuraavassa tietomallipalvelimelle lähetettävässä IFC-tiedoston versiossa, jos rakenneosan GlobalId-tunniste säilyy samana.

3.3 Laadulliset vaatimukset

Suorituskyky ja käytettävyys. Tietomallipalvelimen toimintojen tulee olla sen verran nopeita, että niiden hitaus ei rajoita sovelluksen käyttöä. Palvelimelle lähetettyihin pyyntöihin tulisi vastata alle viidessä sekunnissa. Alkuvaiheessa yhtaikaisten käyttäjien määrä on enimmillään alle sata. Järjestelmä on kuitenkin toteutettava käyttäen sellaisia tekniikoita, että sitä pystytään skaalaamaan tarvittaessa, jotta voidaan palvella useampiakin yhtaikaisia käyttäjiä.

Ylläpitotoimet. Tietomallipalvelin ei ilmoita levytilan loppumisesta. Tietomallipalvelimen ylläpitäjän tulee huolehtia levytilan riittävyydestä sekä tiedostojen poistamisesta tietomallipalvelimelta levytilan loppuessa. Tietomallipalvelin ei osaa toipua palvelinympäristön tai tietokannanhallintajärjestelmän kaatumisesta. Näissä tapauksissa tietomallipalvelimen ylläpitäjän on käynnistettävä kaatuneet prosessit uudelleen manuaalisesti.

Saavutettavuus ja virheistä toipuminen. Tietomallipalvelimen tulisi olla käytettävissä ainakin arkisin ja virka-aikana. Tietomallipalvelimen tulisi ilmoittaa virhetilanteista aina kun mahdollista. Esimerkiksi virheestä tietokantayhteydessä tulisi ilmoittaa käyttäjälle käyttöliittymässä.

Varmuuskopiointi ja palautettavuus. Tietomallipalvelimen tietokanta tulisi varmuuskopioida säännöllisin väliajoin. Järjestelmän ei kuitenkaan tarvitse toteuttaa varmuuskopiointia, mutta tietokanta on valittava niin, että siitä löytyy varmuuskopiointimahdollisuus. Varmuuskopio tulisi olla palautettavissa niin, että järjestelmän ei tarvitse olla poissa käytöstä virka-aikana

3.4 Rajoitukset ja reunaehdot

Ohjelmistorajoitteet. A-Insinöörit Suunnittelu Oy:ssä olemassa oleva palvelinympäristö käyttää Microsoft Server-käyttöjärjestelmää. Tällä hetkellä käytössä olevat versiot ovat

2012 ja 2008 R2. Toteutettavan tietomallipalvelimen tulee olla yhteensopiva näiden käyttöjärjestelmien kanssa ja olla asennettavissa näihin käyttöjärjestelmiin. Tällä voidaan vähentää palvelimen asennus- ja ylläpitokustannuksia, koska voidaan hyödyntää jo olemassa olevaa palvelinympäristöä.

Yrityksen työntekijöillä on käytössä Microsoftin Windows-käyttöjärjestelmä. Käytössä olevat versiot ovat tällä hetkellä 7 ja 8.1. Tietomallipalvelimen käyttöliittymän tulee toimia näissä käyttöjärjestelmän versioissa.

Standardit. Tietomallipalvelimelle lähetetyt tietomallit ovat IFC-standardin mukaisia. Tietomallipalvelimen täytyy tukea versioita IFC2x3 TC1 ja IFC4 (IFC2x3 TC1 Specification, IFC4 Specification). Lähetettävät tietomallit on tallennettu käyttäen STEP-rakennetta (IFC Overview summary).

3.5 Liittyvät järjestelmät

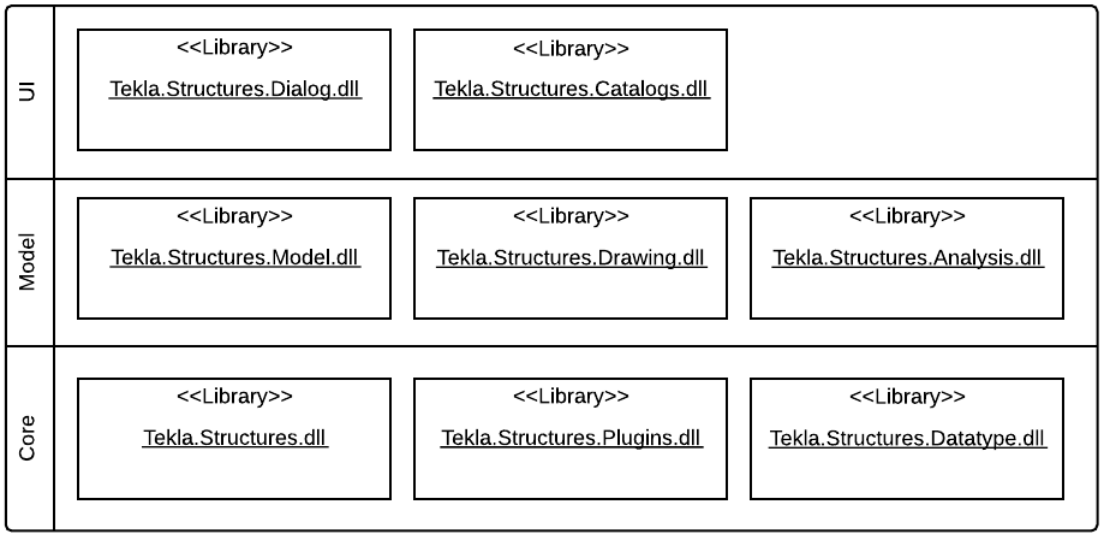
Tietomallipalvelimelle lähetettäviä tietomalleja suunnitellaan A-Insinöörit Suunnittelu Oy:ssa Tekla Structures-suunnitteluohjelmistolla (Tekla Structures). Suunnitteluohjelmisto tarjoaa toiminnot, joilla rakennuksen tietomallin voi suunnitella 3D-näkymässä. Luodusta tietomallista ja sen rakenneosista on myös mahdollista luoda 2D-piirustuksia. Tekla Structures tallentaa tietomallin omassa binäärimuotoisessa formaatissa, mutta sisältää toiminnon myös IFC-tiedostojen luomiseen.

Tekla Structures tarjoaa myös Open API-rajapinnan, jonka avulla on mahdollista ohjelmoida omia liitännäisiä tai laajennoksia, jotka toimivat vuorovaikutuksessa isäntäsovelluksen kanssa. Open API-rajapinnan avulla luodaan laajennos, joka kommunikoi tietomallipalvelimen kanssa. Laajennosta käytetään tietomallipalvelimella olevan varmuuskopion palauttamiseen Tekla Structures-suunnitteluohjelmiston tietomalliin.

Liitännäiset ovat DLL-kirjastoja, jotka ladataan Tekla Structures-suunnitteluohjelmiston käynnistyksen yhteydessä ja ne suoritetaan sen kanssa samassa prosessissa. Laajennokset ovat .exe-tiedostoja, joita suoritetaan erillisessä prosessissa, kun Tekla Structures on käynnissä. (Tekla Open API Developer's Guide)

Open API-rajapinta on toteutettu Microsoftin .NET-ohjelmistokehyksen avulla. Liitännäisten ja laajennosten ohjelmointiin voi käyttää C#- tai VBA-ohjelmointikieliä (Tekla Open API Developer's Guide). Näistä suositellaan käytettäväksi kuitenkin C#-ohjelmointikieltä, koska kaikki virallinen tukimateriaali ja rajapintakuvauksen yhteydessä olevat koodiesimerkit on kirjoitettu sillä. Käyttöliittymän toteutukseen voi käyttää Teklan kehittämää Input File Format -määrittelyä tai Windows Forms-teknologiaa. (Tekla Open API Developer's Guide) Käyttöliittymän toteutukseen voi halutessaan käyttää muitakin teknologioita, mutta tällöin ei voi hyödyntää rajapinnan sisältämiä valmiita käyttöliittymäkomponentteja.

Rajapinta koostuu erilaisista kirjastoista, jotka sisältävät luokat ja metodit suunnitteluohjelmistossa auki olevan tietomallin muokkaamiseen. Rajapinta tarjoaa lähes kaikki samat ominaisuudet tietomallin luontiin ja muokkaukseen kuin Tekla Structures-sovelluksen käyttöliittymäkin. Alla olevassa kuvassa (Kuva 3.2) näkyvät Open API-rajapinnan sisältämät kirjastot. Kirjastot voidaan jakaa kolmeen osioon niiden sisältämien toimintojen perusteella. Core-osiosta löytyy kirjastot, joita käytetään hyväksi muissa osioissa. Model-osion kirjastoja käytetään suunnittelusovelluksessa olevan tietomallin muokkaukseen. Model-osiosta löytyvät myös luokat, joilla esitetään tietomallissa olevia rakenneosia. UI-osion kirjastoja käytetään käyttöliittymän luonnissa.



Kuva 3.2 Tekla Open API-kirjastot.

Tekla Structures-kirjasto pitää sisällään luokkia, joita käytetään hyväksi Tekla Structures Model- ja Tekla Structures Drawing-kirjastoissa. Kirjastosta löytyy tämän lisäksi luokkia, joiden avulla voidaan tutkia Tekla Structures-suunnitteluohjelmiston asetuksia ja ympäristömuuttujia. (Tekla Open API Developer's Guide)

Tekla Structures Plugins-kirjasto sisältää luokat ja metodit, joita käytetään hyväksi liitännäisten luonnissa. Kirjastosta löytyvät abstraktit kantaluokat joista kaikki liitännäiset tulee periyttää. Kaikki tietomallia muokkaavat liitännäiset tulee periyttää PluginBase-luokasta ja piirustuksia muokkaavat liitännäiset DrawingPluginBase-luokasta. Samassa luokassa määritellään myös ne metodit, jotka jokaisen liitännäisen tulee toteuttaa.

Tekla Structures Datatype-kirjasto pitää sisällään tietotyyppejä, joita käytetään hyväksi muissa kirjastoissa olevissa luokissa (Tekla Open API Developer's Guide). Kirjastosta löytyy esimerkiksi Distance-luokka, jonka avulla voidaan kuvata eri mittayksikköjärjestelmien pituusyksiköitä.

Tekla Structures Model-kirjasto sisältää luokat, joilla kuvataan tietomallissa olevia rakenneosia. Rakenneosia kuvaavat luokat ja niiden sisältämät jäsenmuuttujat on pyritty

nimeämään samalla tavalla kuin Tekla Structures-sovelluksen käyttöliittymässäkin (Tekla Open API Developer's Guide). Rakenneosia kuvaavat luokat sisältävät myös metodit, joilla kyseisiä rakenneosia voi luoda, muokata tai poistaa. Kirjasto sisältää myös koko tietomallin muokkaukseen tarkoitetut metodit, joiden avulla esimerkiksi kaikkia tietomallin sisältämiä osia voi hakea ja muokata.

Tekla Structures Drawing-kirjasto sisältää luokkia, joilla kuvataan tietomallin rakenneosista tehtyjä piirustuksia (Tekla Open API Developer's Guide). Luokat sisältävät metodeja, joilla tietomallissa olevia piirustuksia voi muokata.

Tekla Structures Analysis-kirjasto sisältää analyysimallien luontiin ja muokkaukseen tarkoitetut luokat ja metodit (Tekla Open API Developer's Guide). Analyysimallien avulla voidaan kuvata esimerkiksi rakenneosiin kohdistuvia kuormia.

Tekla Structures Dialog-kirjasto pitää sisällään luokat, joita käytetään käyttöliittymien luonnissa (Tekla Open API Developer's Guide). Kirjastosta löytyy PluginFormBase-kantaluokka, josta Windows Forms-teknologialla toteutetut käyttöliittymät tulee periyttää. Kirjasto sisältää myös luokat, joita käytetään käyttöliittymätekstien lokalisoinnissa.

Tekla Structures Catalogs-kirjasto sisältää erilaisia käyttöliittymäkomponentteja, joita voidaan hyödyntää rajapinnan avulla luotujen liitännäisten ja laajennosten käyttöliittymissä (Tekla Open API Developer's Guide). Kirjastosta löytyvät myös luokat joiden avulla Tekla Structures-suunnitteluosovelluksen sisältämiä profiili- ja materiaalikirjastoja voi hakea.

Seuraavalla sivulla olevassa kuvassa (Kuva 3.3) on C#-ohjelmointikielellä toteutettu koodiesimerkki, jossa esitellään Tekla Open API:n toimintaa. Esimerkissä näytetään kuinka rajapinnan avulla voidaan luoda palkki Tekla Structures tietomalliin.

Koodiesimerkissä luodaan ensin `Model`-luokan olio, joka kuvaa Tekla Structures tietomallia. Tämän jälkeen tarkistetaan onko yhteys Tekla Structures tietomalliin olemassa `Model`-luokan sisältämän `GetGonnectionStatus`-metodin avulla. Metodi palauttaa `true`, siinä tapauksessa, kun suunnitteluosovellus on auki ja yhteys tietomalliin saadaan luotua. Muussa tapauksessa, metodi palauttaa `false`. Metodin käyttö on suositeltavaa aina ennen varsinaisen toimintalogiikkaa sisältävän koodin suorittamista.

Tämän jälkeen luodaan palkkia kuvaava `Beam`-olio. `Beam`-oliolle asetetaan sen tarvitsemat attribuuttien arvot. Palkille asetetaan esimerkiksi sen alku- ja loppupiste sen profiili sekä materiaali.

Lopuksi palkki lisätään tietomalliin `Beam`-olion sisältämän `Insert`-metodin avulla. `Insert`-metodi palauttaa `true`, siinä tapauksessa jos palkin lisäys tietomalliin onnistui. Jos palkin lisäys ei onnistu, metodi palauttaa arvon `false`. Lisäyksen epäonnistuminen voi

johtua esimerkiksi palkille asetetuista virheellisistä attribuuttiarvoista tai yhteyden katkeamisesta tietomalliin.

```
public bool CreateBeam()
{
    bool insertionResult = false;

    // Luodaan malli olio
    Model model = new Model();

    // Tarkistetaan onko yhteys Tekla Structures tietomalliin olemassa
    bool connectedToTs = model.GetConnectionStatus();

    if (!connectedToTs)
        return insertionResult;

    // Luodaan palkki olio
    Tekla.Structures.Model.Beam beam = new Tekla.Structures.Model.Beam();
    Point startPoint = new Point(0, 0, 0); // Palkin alkupiste (x, y, z)
    Point endPoint = new Point(1000, 0, 0); // Palkin loppupiste (x, y, z)
    beam.StartPoint = startPoint;
    beam.EndPoint = endPoint;
    beam.Profile.ProfileString = "IPE200"; // Palkin profiili
    beam.Material.MaterialString = "S355J2"; // Palkin materiaali

    // Luodaan palkki tietomalliin
    insertionResult = beam.Insert();

    return insertionResult;
}
```

Kuva 3.3 Tekla Open API-koodiesimerkki palkin luonnista tietomalliin.

4. VALITUT TOTEUTUSTEKNIIKAT

Tässä luvussa esitellään tietomallipalvelimen toteutustekniikat. Ensin esitellään yleisellä tasolla mihin valittuja toteutustekniikoita käytetään järjestelmän toteutuksessa. Seuraavaksi käydään läpi tietomallipalvelimen toteutuksessa käytettävät sovelluskehykset. Tämän lisäksi esitellään sovelluskehystä laajentavat kolmannen osapuolen kirjastot sekä käytettävä ohjelmointikieli ja kehitysympäristö. Tämän jälkeen tarkastellaan käyttöliittymän toteutukseen käytettäviä tekniikoita. Lopuksi esitellään tietomallipalvelimen taustalla käytettävä tietokanta.

4.1 Järjestelmän yleiskuvaus

Tietomallipalvelin toteutetaan web-sovelluksena, jota käytetään verkkoselaimella. Verkkoselaimen ja web-sovelluksen väliseen viestintään käytetään HTTP-protokollaa (IETF 2014a).

4.1.1 Yleiskuvaus toteutustekniikoista

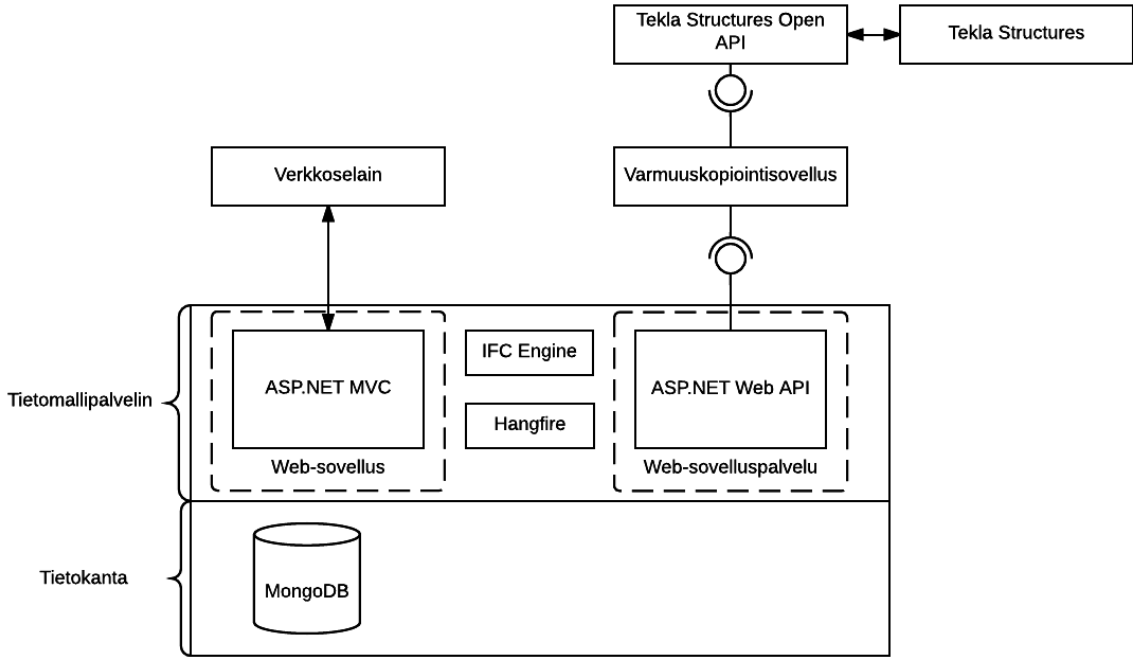
Seuraavalla sivulla kuvassa (Kuva 4.1) on esitelty tietomallipalvelimen toteutustekniikat yleisellä tasolla. Web-sovelluksen toteuttamisessa käytetään Microsoftin ASP.NET MVC-sovelluskehystä. Sovelluskehystä päätettiin käyttää tässä projektissa, koska sen käytöstä on yrityksessä jo aiempaa kokemusta ja sen avulla on toteutettu yrityksessä käytössä olevia intranet-sovelluksia. Tästä syystä A-Insinöörit Suunnittelu Oy:ssä on myös olemassa palvelinympäristö, johon kyseisen sovelluskehyksen avulla toteutetut web-sovellukset voidaan asentaa.

Tietomallipalvelimen toteutuksessa hyödynnetään kolmannen osapuolen kirjastoja, joilla laajennetaan sovelluskehyksen toiminnallisuutta. Tietomallipalvelimella suoritettavien pitkäaikaisten tehtävien suorittamisessa käytetään Hangfire-kirjastoa. Tietomallipalvelimelle lähetettyjen IFC-tiedostojen parsintaan ja käsittelyyn käytetään IFC Engine-kirjastoa.

Tietomallipalvelimelle lähetettyjen tietomallien säilyttämiseen käytetään MongoDB-tietokantaa. Tietokantaan tallennetaan myös projektien rakenne, ja rakenneosille linkitetyt dokumentit sekä käyttäjätiedot.

Tietomallipalvelimen käyttöliittymä toteutetaan HTML:n avulla. Näkymien dynaamisessa päivittämisessä käytetään apuna jQuery-kirjastoa, joka sisältyy ASP.NET MVC-sovelluskehykseen. Tietomallipalvelimen tietomallien 3D-tarkastelunäkymän toteutukseen käytetään HTML-elementtien lisäksi WebGL-teknologiaa. Näkymän toteutuksessa

käytetään apuna Three.js-kirjastoa, joka abstrahoi WebGL:n tarjoaman rajapinnan ja tarjoaa monia valmiita toimintoja, joilla voidaan nopeuttaa sovelluskehitystä.



Kuva 4.1 Tietomallipalvelimen yleiskuvaus.

Tietomallipalvelimen toiminnallisena vaatimuksena oleva rajapinta toteutetaan web-sovelluspalveluna. Web-sovelluspalveluilla tarkoitetaan ohjelmien käyttöön suunniteltuja web-sovelluksia, jotka tarjoavat ohjelmalliseen käyttöön soveltuvan rajapinnan. Web-sovelluspalvelut eroavat normaalista web-sovelluksista siinä, että ne lähettävät asiakkaalle HTML-sivujen sijasta dataa esimerkiksi XML-muodossa, koska se on helpompi käsitellä ohjelmallisesti.

Rajapintaa hyödynnetään tietomallipalvelimen varmuuskopiointisovelluksessa, jota käytetään tietomallipalvelimella olevien tietomallien palauttamiseen Tekla Structures-suunnitteluohjelmistoon. Varmuuskopiointisovellus toteutetaan työpöytäsovelluksena, joka kommunikoi Tekla Structures-suunnitteluohjelmiston kanssa Tekla Open API-rajapinnan avulla.

Web-sovelluspalvelun toteutuksessa käytetään Microsoftin kehittämää ASP.NET Web API-sovelluskehystä. Web-sovelluspalvelun toteutuksessa pyritään noudattamaan REST-arkkitehtuurityyliä.

4.1.2 REST

REST eli Representational State Transfer on arkkitehtuurityyli hajautettujen hypermediajärjestelmien toteuttamiseen. REST:n kehittäjä Roy Fielding esitteli arkkitehtuurityylin ensimmäistä kertaa väitöskirjassaan vuonna 2000. (Fielding 2000)

REST arkkitehtuurityyli soveltuu hyvin web-sovellusten tarjoamien rajapintojen toteuttamiseen HTTP-protokollan avulla. REST arkkitehtuurityyliä ei ole kuitenkaan sidottu pelkästään web-sovelluksiin ja HTTP-protokollaan, vaan sitä voidaan hyödyntää myös muualla.

REST määrittelee viisi pakollista rajoitetta ja yhden vapaaehtoisen rajoitteen, joita toteuttavan järjestelmän tulisi noudattaa. Rajoitteiden avulla pyritään mm. parantamaan järjestelmän skaalautuvuutta ja yhtenäistämään järjestelmän tarjoama rajapinta. Rajoitteiden tavoitteena on myös vähentää asiakassovelluksen ja palvelimen välisen verkkoliikenteen määrää sekä tehostaa järjestelmän tietoturvaa. (Fielding 2000, p. 75)

Asiakas-palvelin arkkitehtuuri. Ensimmäinen rajoite on asiakas-palvelin-arkkitehtuurin hyödyntäminen. Palvelimen tehtävänä on tarjota palveluita ja vastaanottaa asiakkaan lähettämiä pyyntöjä. Asiakas voi lähettää palvelimelle pyynnön haluamaansa palveluun, jolloin palvelin suorittaa kyseisen palvelun vaatimat toimenpiteet ja palauttaa vastauksen asiakkaalle. Palvelin voi myös kieltäytyä pyynnöstä ja lähettää asiakkaalle vain kieltäytymisen syyn. Asiakas-palvelin arkkitehtuuria hyödyntämällä voidaan erottaa käyttöliittymä ja tietovarasto toisistaan, joka parantaa järjestelmän skaalautuvuutta. (Fielding 2000, p. 45, 78)

Tilattomuus. Toisena rajoitteena on tilattomuus. Asiakkaan ja palvelimen välisen kommunikaation täytyy olla tilatonta. Jokaisen asiakkaan palvelimelle lähettämän pyynnön mukana tulee olla kaikki sen käsittelemiseen tarvittava tieto. Palvelimella ei tule siis säilyttää tietoja asiakkaan aiemmin lähetyistä pyynnöistä, vaan se jää asiakassovelluksen tehtäväksi. Tällä voidaan parantaa esimerkiksi järjestelmän skaalautuvuutta ja toipua paremmin palvelimen kaatumisesta. (Fielding 2000, p. 78-77)

Välimuistin käyttö. Kolmas rajoite on välimuistin käyttö. Palvelimen lähettämän vastauksen mukana voidaan kertoa, onko kyseisen vastauksen sisältämä data mahdollista tallentaa asiakassovelluksen välimuistiin vai tuleeko asiakkaan hakea data aina uudelleen palvelimelta, jotta voidaan varmistua sen ajantasaisuudesta. Välimuistia hyödyntämällä voidaan vähentää asiakkaan ja palvelimen välistä verkkoliikennettä. (Fielding 2000, p. 79-80)

Yhtenäinen rajapinta. Neljäs rajoite on yhtenäinen rajapinta. Kaikkien järjestelmän komponenttien välillä tulee olla yhtenäinen rajapinta. Yhtenäisen rajapinnan avulla voidaan varmistaa komponenttien yhteensopivuus sekä yksinkertaistaa järjestelmän arkkitehtuuria ja vähentää komponenttien välisiä riippuvuuksia. (Fielding 2000, p. 81-82)

Yhtenäisen rajapinnan toteuttamiseksi Fielding (2000, p. 81-82) määrittelee neljä vaatimusta:

- Resurssien tunnistaminen. Resurssi on yksi REST arkkitehtuurin keskeisimmistä käsitteistä. Sillä tarkoitetaan mitä tahansa informaatiota, jolle voidaan antaa yksikäsitteinen tunniste. Resurssi voi olla esimerkiksi dokumentti, kuva tai kokoelma muista resursseista. (Fielding 2000, p. 88)

- Resurssien muokkaus niiden esitysten kautta. Resurssilla tulee olla esitys, joka koostuu resurssin sisältämästä datasta ja sitä kuvailevasta metadatasta. Resurssien esityksessä tulisi käyttää standardeja mediatyyppejä, kuten esimerkiksi HTML tai XML. Jotta yhtenäinen rajapinta toteutuisi, resurssien muokkaukseen ja käsittelyyn tulee käyttää standardoituja metodeja. Resurssien muokkauksessa käytetään hyväksi niiden esityksiä. Esimerkiksi uutta resurssia luotaessa asiakkaan tulee lähettää viestin mukana esitys uudesta resurssista palvelimelle. (Fielding 2000, p. 91, 99)

- Itsensä kuvaavat viestit. Viestien tulee sisältää metadataa, jonka avulla asiakas tai palvelin voi päätellä viestin tarkoituksen, kuten esimerkiksi suoritettavan metodin tai operaation onnistumisen. Itsensä kuvaavat viestit liittyvät myös tilattomuuden rajoitteeseen, sillä niiden avulla voidaan kuvata pyynnön käsittelemiseksi tarvittava tieto. Metadataa voidaan käyttää myös välimuistirajoitteen toteuttamiseen. (Fielding 2000, p. 91)

- Hypermedian käyttö sovelluksen tilan säilyttämiseen. Palvelimen palauttama resurssin esitys voi sisältää tiedon siitä mitä muita resursseja kyseiseen resurssiin liittyy ja mitä toimintoja sille voidaan suorittaa. Näiden tietojen avulla asiakas voi siirtyä toiseen tilaan sovelluksessa, eikä sovelluksen tilaa tarvitse säilyttää palvelimella. Tällä rajoitteella autetaan myös tilattomuus rajoitteen toteuttamista.

Kerrosarkkitehtuuri. Viidentenä rajoitteena on kerrosarkkitehtuurin hyödyntäminen. Kerrosarkkitehtuurin mukainen järjestelmä koostuu tasoista, jotka on järjestetty jonkin asbstrahointiperiaatteen mukaisesti nousevaan järjestykseen. Alemman tason palvelut tarjoavat palveluja ylemmille kerroksille. Yleensä alimmalla tasolla ovat laitteistoa lähimpänä olevat palvelut, kuten tietokannan kanssa kommunikoivat komponentit ja ylemmällä tasolla asiakassovelluksen kanssa kommunikoivat komponentit. Kerrosarkkitehtuuri mahdollistaa esimerkiksi välityspalvelimen käytön, jolla voidaan tasata asiakkaalta tulevia pyyntöjä useammille palvelimille. Kerrosarkkitehtuurin avulla voidaan myös kapseloida vanhempia sovelluksen osia, jotta ne ovat käytettävissä myös uusilla asiakassovelluksilla. (Fielding 2000, p.46, p. 82-83; Koskimies & Mikkonen 2005, s. 126)

Code-on-demand. Kuudes rajoite on Code-on-demand. Tällä tarkoitetaan sitä, että asiakassovellus voi tarvittaessa laajentaa toiminnallisuuttaan lataamalla palvelimelta ohjelmakoodia, jonka se suorittaa. Tämä rajoite on vaihtoehtoinen, eikä kaikkien REST-arkkitehtuurin mukaisten järjestelmien ole pakko toteuttaa tätä rajoitetta. (Fielding 2000, p. 84)

Code-on-demand-rajoitteen tarkoituksena on parantaa asiakassovelluksen laajennettavuutta. Rajoite mahdollistaa sen, että kaikkia asiakassovelluksen toimintoja ei tarvitse toteuttaa etukäteen, vaan niitä voidaan lisätä myöhemmin dynaamisesti tarvittaessa. (Fielding 2000, p. 84-85)

4.1.3 REST-arkkitehtuurityylin soveltaminen web-sovelluksissa

Koska REST on arkkitehtuurityyli, se ei suoraan määrittele, kuinka sitä hyödyntävät sovellukset tulisi toteuttaa käytännössä. Useimmat REST-arkkitehtuurityylissä määritetyt rajoitteet ovat kuitenkin helposti toteutettavissa web-sovellusten rajapintoja luotaessa.

Asiakas-palvelin rajoite toteutuu web-sovelluksissa, sillä ne on toteutettu yleensä asiakas-palvelin arkkitehtuuria hyödyntäen. Usein web-sovelluksissa hyödynnetään myös kerrosarkkitehtuuria, jolloin alimmalla tasolla on esimerkiksi tietokanta, joka sisältää resurssien sisältämän datan ja ylimmällä tasolla palvelin, joka kommunikoi asiakkaan kanssa.

Tilattomuuden rajoite toteutuu myös web-sovelluksissa, koska useimmiten asiakkaan ja palvelimen väliseen kommunikointiin käytetään HTTP-protokollaa, joka on tilaton (IETF 2014a, p. 1). Rajoitteen toteuttamiseksi tulee vain huolehtia siitä, että asiakkaan tilaa ei tallenneta palvelimelle ja, että asiakas lähettää HTTP-pyynnön mukana kaikki sen suorittamiseksi tarvittavat tiedot.

Välimuistin rajoite on toteutettavissa web-sovelluksissa hyödyntämällä HTTP-protokollan ominaisuuksia. Rajoitteen toteuttamiseksi HTTP-protokollan otsikkotiedoissa voidaan ilmoittaa, onko kyseinen data mahdollista tallentaa myöhempää käyttöä varten, vai tuleeko data aina hakea uudelleen. (IETF 2014b, p. 21)

Web-sovelluksissa yhtenäisen rajapinnan rajoitteen toteuttamisessa voidaan hyödyntää URI-osoitteita (NWG 2005). Tällöin jokaiselle resurssille annetaan URI-osoite, joka on resurssin yksikäsitteinen tunniste. Resurssien sisältämän tiedon hakemisessa ja muokkauksessa voidaan käyttää HTTP-protokollan tarjoamia standardeja metodeja. Metodeja käytetään myös itsensäkuvaavien viestien rajoitteen toteuttamisessa. Asiakas voi ilmaista metodien avulla operaation tarkoituksen HTTP-pyynnön otsikkotiedoissa.

Rajapinnan toteutuksessa hyödynnetään yleisimmin metodeja GET, POST, PUT, DELETE ja OPTIONS. HTTP määrittelee osan metodeista, kuten GET, PUT, DELETE ja OPTIONS idempotenteiksi. Idempotenttien metodien tulisi tuottaa aina saman lopputuloksen riippumatta siitä kuinka monta kertaa ne suoritetaan, jos ne kohdistuvat samaan

resurssiin. Osa pyynnöistä, kuten GET ja OPTIONS ovat myös turvallisia. Turvalliset pyynnöt eivät saa muokata kyseisen resurssin tilaa. (IETF 2014a, p. 22-23)

Metodeille ennalta määriteltyä semantiikkaa voidaan hyödyntää rajapinnan luomisessa. Tällöin asiakas ja palvelin voivat molemmat olla varmoja metodin käyttötarkoituksesta. Alla olevassa taulukossa (Taulukko 4.1) on listattu HTTP-pyynnön metodien käyttötarkoitus REST-arkkitehtuurityylin mukaisissa web-sovelluksissa.

Taulukko 4.1 HTTP-metodeiden semantiikka ja käyttötarkoitus REST-arkkitehtuurityylin mukaisissa web-sovelluksissa.

HTTP-metodi	Käyttötarkoitus	Turvallinen (Safe)	Idempotentti
GET	Resurssin hakeminen	Kyllä	Kyllä
POST	Resurssin luominen	Ei	Ei
PUT	Resurssin päivittäminen	Ei	Kyllä
DELETE	Resurssin poistaminen	Ei	Kyllä
OPTIONS	Resurssiin liittyvien operaatioiden hakeminen	Kyllä	Kyllä

Itsensäkuvaavien viestin rajoitteen toteuttamisessa käytetään myös HTTP-vastauksen otsikkotietoja. Palvelin voi ilmaista otsikkotietojen avulla asiakkaalle esimerkiksi pyynnön onnistumisen tai epäonnistumisen käyttämällä HTTP-statuskoodeja. (IETF 2014a, p. 47)

Varsinainen resurssin esitystä kuvaava data voidaan kuljettaa HTTP-pyynnön ja vastauksen rungossa. Datan tyyppi esitetään HTTP-protokollan tarjoaman `Content-Type`-kentän avulla (IETF 2014a, p. 10). HTTP-vastauksen runkoa on mahdollista käyttää myös sovelluksen tilan säilyttämiseen, jolloin siinä voidaan esittää resurssiin liittyvät operaatiot.

Valinnainen Code-on-demand rajoite voidaan toteuttaa web-sovelluksissa esimerkiksi JavaScriptilla. JavaScriptin avulla voidaan luoda sovellus, joka ladataan palvelimelta ja suoritetaan asiakkaan selaimessa ajonaikaisesti. Koska sovellus ladataan palvelimelta, voidaan sitä myös tarvittaessa päivittää ja laajentaa sen toiminnallisuutta. Tällöin tulee vain huolehtia siitä, että asiakas ei tallenna sovellusta välimuistiin, vaan lataa sen aina uudelleen palvelimelta käytön yhteydessä.

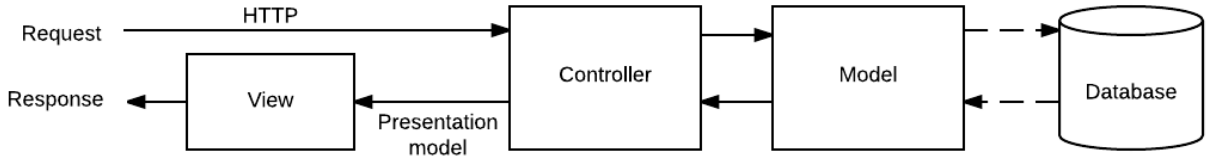
4.2 Sovelluskehys

Tietomallipalvelimen toteutuksessa hyödynnetään kahta eri sovelluskehystä. ASP.NET MVC-sovelluskehystä käytetään tietomallipalvelimen toteutuksessa ja ASP.NET Web API-sovelluskehyksen avulla toteutetaan tietomallipalvelimen rajapinta. Tämän lisäksi toteutuksessa hyödynnetään kolmannen osapuolen kirjastoja, joilla laajennetaan sovelluskehyksen toiminnallisuutta. Seuraavissa aliluvuissa on esitelty tarkemmin käytettävät sovelluskehykset sekä kolmannen osapuolen kirjastot. Tämän lisäksi esitellään käytettävä kehitysympäristö ja ohjelmointikieli.

4.2.1 ASP.NET MVC

ASP.NET MVC on Microsoftin kehittämä web-sovellusten kehitykseen tarkoitettu sovelluskehys. Sovelluskehys hyödyntää Microsoftin .NET-sovelluskehystä ja sen ydin perustuu ASP.NET-teknologiaan. (Ciliberti 2013, Ch. 2)

ASP.NET MVC-sovelluskehyksen arkkitehtuuri perustuu MVC-suunnittelumalliin. Suunnittelumallin lyhenne MVC tulee sanoista Model, View ja Controller. Suunnittelumallin ideana on erottaa toisistaan malli (Model), näkymä (View) ja ohjain (Controller). Suunnittelumallin sisältämät osat ja niiden väliset suhteet on esitetty alla olevassa kuvassa (Kuva 4.2).



Kuva 4.2 ASP.NET MVC-suunnittelumalli (perustuu lähteeseen Freeman 2014b).

Malli. Mallin tehtävänä on säilyttää sovelluksen tarvitsema tietosisältö. ASP.NET MVC-sovelluskehys voi käyttää mallin luokkana mitä tahansa .NET-luokkaa. Mallin tietosisältö voidaan kuvata luokan ominaisuuksien avulla. Malli voi olla myös monimutkaisempi ja sisältää tiedon lisäksi toiminnallisuuksia, kuten validointia. Tällä voidaan varmistua siitä, että mallin kuvaama tietosisältö säilyy eheänä. Luokan sisältämien ominaisuuksien validoinnin apuna voidaan käyttää Data annotations-attribuutteja. Validointia käytetään hyväksi HTML-lomakkeissa. Sovelluskehys osaa muodostaa automaattisesti yhteyden ominaisuuksien ja lomakkeen välillä, kun näkymän luonnissa käytetään HtmlHelper-luokan tarjoamia metodeja. (Ciliberti 2013, Ch. 2; Freeman 2014b, Ch. 2; ASP.NET MVC Overview)

Seuraavan sivun kuvassa (Kuva 4.3) on näkyvissä esimerkki C#-ohjelmointikielellä toteutetusta malliluokasta. Malliluokan tietosisältö on kuvattu luokan ominaisuuksien avulla. Ominaisuuksien validoinnissa on käytetty hyväksi Data annotations-attribuutteja. `Required`-attribuutin avulla voidaan kertoa kyseisen ominaisuuden olevan pakollinen. Attribuutille voidaan määrittää virheviesti, joka näytetään siinä tapauksessa, kun kyseisen ominaisuuden kenttä HTML-lomakkeessa jätetään tyhjäksi. `StringLength`-attribuutin avulla voidaan rajoittaa `string`-tyyppisen luokan ominaisuuden minimi- ja maksimipituus. `EmailAddress`-attribuutin avulla voidaan tarkistaa onko kyseisen ominaisuuden sisältämä arvo sähköpostiosoitteen mukaisessa formaatissa. `HiddenInput`-attribuutin avulla voidaan määritellä ominaisuudet, joita ei haluta näkyviin HTML-lomakkeessa. `DisplayName`-attribuutin voidaan määrittää kyseiselle ominaisuudelle käyttöliittymässä näytettävä teksti.

```
public class PersonModel
{
    [HiddenInput(DisplayValue = false)]
    public int Id { get; set; }

    [Required(ErrorMessage="Käyttäjätunnus on pakollinen")]
    [StringLength(15, ErrorMessage = "Käyttäjätunnuksen täytyy olla 3-15 merkkiä pitkä", MinimumLength = 3)]
    [DisplayName("Käyttäjätunnus")]
    public string UserName { get; set; }

    [Required(ErrorMessage = "Salasana on pakollinen")]
    [StringLength(50, ErrorMessage = "Salasanan täytyy olla 8-50 merkkiä pitkä", MinimumLength = 8)]
    [DisplayName("Salasana")]
    public string Password { get; set; }

    [DisplayName("Etunimi")]
    public string FirstName { get; set; }
    [DisplayName("Sukunimi")]
    public string LastName { get; set; }

    [Required(ErrorMessage = "Sähköposti on pakollinen")]
    [EmailAddress]
    [DisplayName("Sähköposti")]
    public string Email { get; set; }
}
```

Kuva 4.3 Esimerkki ASP.NET MVC-sovelluskehyksen mallista.

Mallia käytetään myös abstraktiokerroksena tietokantaan. Malli on siis yleensä myös yhteydessä tietokantaan ja käyttää sitä tiedon hakemiseen sekä oman tilansa tallentamiseen. (Ciliberti 2013, Ch. 2; Freeman 2014b, Ch. 2; ASP.NET MVC Overview)

Näkymä. Näkymien tehtävänä on esittää mallien tilaa käyttöliittymässä. Näkymät mahdollistavatkin mallien tietosisällön sisältävien .NET-luokkien ja niiden ominaisuuksien näyttämisen. Näkymien rakentamisessa käytetään apuna näkymämoottoreita. ASP.NET MVC-sovelluskehys sisältää oletuksena kaksi näkymämoottoria, jotka ovat Web Forms ja Razor. Tällä hetkellä näkymämoottoreista käytetympi on Razor ja sitä on tarkoitus hyödyntää myös tietomallipalvelimen toteutuksessa. (Ciliberti 2013, Ch. 2; Freeman 2014b, Ch. 2)

Näkymämoottoreilla on oma merkkauskielensä, jonka avulla voidaan luoda sivupohjia. Merkkauskieli sisältää mm. muuttujat, ehtorakenteet ja silmukat, joiden avulla voidaan ohjata käyttöliittymän rakentamisen logiikkaa. Razor tukee myös C#- tai VB.NET-ohjelmointikielellä luotuja koodilohkoja, jotka suoritetaan palvelimella. Näkymiin ei kuitenkaan ole tarkoitus rakentaa muuta toiminnallisuutta, kuin se mitä tarvitaan mallien esittämiseen. Näkymämoottoreissa voidaan hyödyntää myös HtmlHelper-luokkaa, jonka avulla voidaan luoda HTML-elementtejä ilman HTML-koodin kirjoittamista ja helpottaa näin sivupohjan rakentamista. Näkymät käyttävät sisällön dynaamisessa päivittämisessä apuna jQuery-kirjastoa, joka on osa ASP.NET MVC-sovelluskehystä. jQuery-kirjastosta kerrotaan tarkemmin kohdassa 4.3.1. (Ciliberti 2013, Ch. 2; Freeman 2014b, Ch. 2)

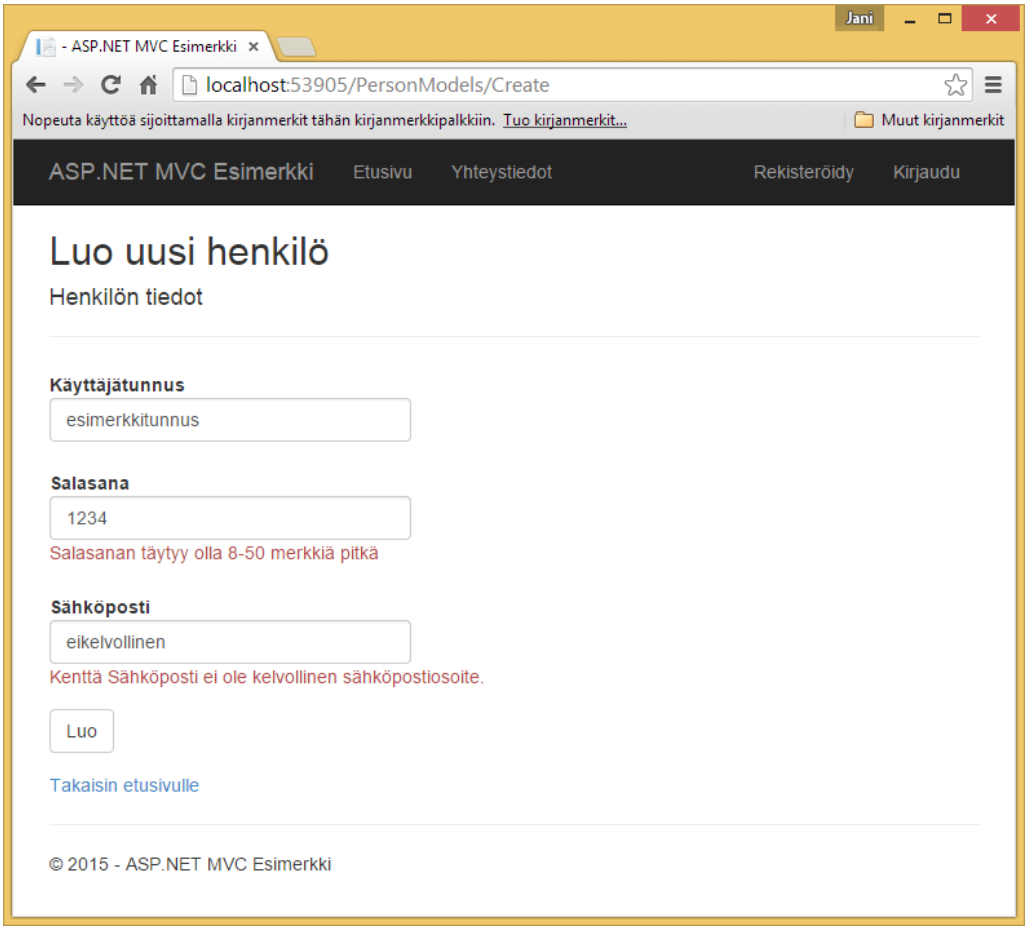
Seuraavan sivun kuvassa (Kuva 4.4) on näkyvissä Razor-merkkauskielellä luotu HTML-lomake. Razor-merkkauskielessä koodilohkot aloitetaan @-merkillä. Esimerkiksi näkymän hyödyntämä malliluokka on määritelty ensimmäisellä rivillä @model-syntaksilla.

```
@model ASPNETMVC.Models.PersonModel

<h2>Luo uusi henkilö</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>Henkilön tiedot</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.UserName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.UserName, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.UserName, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Password, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Password, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Password, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Email, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Email, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Email, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Luo" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Takaisin etusivulle", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Kuva 4.4 Esimerkki Razor-merkkauskielellä luodusta HTML-lomakkeesta.

Lomakkeen teossa on hyödynnetty HtmlHelper-luokan tarjoamia metodeja. Lomake aloitetaan `BeginForm`-metodin avulla. `EditorFor`-metodin avulla voidaan luoda lomakkeen syötekenttiä käyttämällä mallissa määritetyn luokan ominaisuuksia. `LabelFor`-metodin avulla voidaan näyttää malliluokan ominaisuuden nimi. `ValidationMessageFor`-metodia käytetään virheviestin näyttämisessä, jos syötteen tarkistus epäonnistuu.

Näkymämoottorin suorittaa sivupohjan kääntämisen HTML-muotoon, kun se näytetään käyttöliittymässä. Seuraavalla sivulla olevassa kuvassa (Kuva 4.5) on näkyvissä edellisen sivun kuvan Razor-merkkauskielen luoma HTML-lomake Chrome-selaimessa. Kuvassa näkyvät myös virheviestit salasana- ja sähköposti-kentille, jotka eivät ole läpäisseet syötteiden tarkistusta. Näkymämoottorit mahdollistavatkin erittäin dynaamisten web-sivujen luomisen, jotka näyttävät sovelluksen tietosisällön tilan ja mukautuvat käyttäjän syötteisiin.



Kuva 4.5 Razor-merkkauskielen luoma HTML-lomake Chrome-selaimessa.

Näkymämoottorin tehtävänä on myös toteuttaa `IViewEngine`-rajapinta. Tätä rajapintaa sovelluskehys käyttää hyväkseen oikeiden näkymien löytämisessä. (Ciliberti 2013, Ch. 2; Freeman 2014b, Ch. 2)

Ohjain. Ohjain tarjoaa keinon näkymän ja mallin erottamiseen. Ohjaimen tehtävänä on ottaa vastaan käyttäjän lähettämät syötteet ja pyytää mallia tekemään niiden mukaiset muutokset. Ohjain voi käsitellä käyttäjän syötettä ennenkuin välittää sen eteenpäin mallille. Tämän jälkeen ohjain valitsee näytettävän näkymän ja auttaa välittämään mallin tiedon edelleen näkymälle.

ASP.NET MVC-sovelluskehyksessä käytettävät ohjain luokat periytetään `Controller`-luokasta. `Controller`-luokan julkisia metodeita kutsutaan Action-metodeiksi. Jokaiselle Action-metodille luodaan sovelluskehyksen toimesta URL-osoite, jota voidaan kutsua HTTP-pyynnön avulla. Metodin parametrit voidaan välittää URL-parametreina. Action-metodit palauttavat oletuksena abstraktista `ActionResult`-kantaluokasta periytetyn olion. `ActionResult`-luokasta on periytetty esimerkiksi näkymää kuvaava olio, mutta sen avulla voidaan palauttaa myös JSON-objekti tai tiedosto. (Ciliberti 2013, Ch. 2; Freeman 2014b, Ch. 2)

Alla olevassa kuvassa (Kuva 4.6) on esimerkki ASP.NET MVC-ohjain luokan toteutuksesta. Ohjaimen tarjoamien `ActionResult`-tyyppisten julkisten metodien avulla voidaan luoda uusia `PersonModel`-luokan olioita tai hakea jo olemassa olevat oliot.

```
public class PersonModelsController : Controller
{
    private ApplicationDbContext db = new ApplicationDbContext();

    // GET: http://localhost/PersonModels
    public ActionResult Index()
    {
        return View(db.PersonModels.ToList());
    }

    // GET: http://localhost/PersonModels/Details/5
    public ActionResult Details(int? id)
    {
        if (id == null)
        {
            return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
        }

        PersonModel personModel = db.PersonModels.Find(id);
        if (personModel == null)
        {
            return HttpNotFound();
        }
        return View(personModel);
    }

    // GET: http://localhost/PersonModels/Create
    public ActionResult Create()
    {
        return View();
    }

    // POST: http://localhost/PersonModels/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    public ActionResult Create(PersonModel personModel)
    {
        if (ModelState.IsValid)
        {
            db.PersonModels.Add(personModel);
            db.SaveChanges();
            return RedirectToAction("Index");
        }

        return View(personModel);
    }
}
```

Kuva 4.6 Esimerkki ASP.NET MVC-sovelluskehyksen ohjain-luokasta.

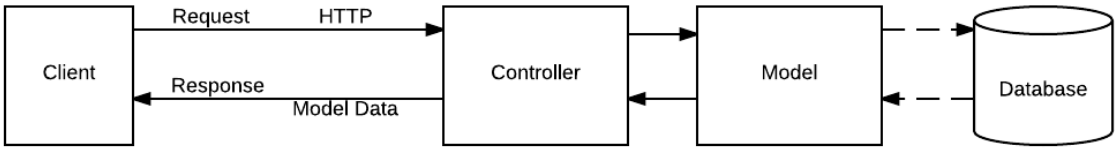
Julkisten metodien yläpuolella näkyy kommenteissa niitä vastaava URL-osoite. Metodit palauttavat `View`-tyyppisen olion, jonka parametrina voidaan välittää malliluokan olioita. Sovelluskehys hoitaa näkymää kuvaavan olion välittämisen näkymämoottorille. Metodeista voidaan palauttaa myös HTTP-statuskoodeja, joilla voidaan ilmaista tietyn operaation epäonnistuminen tai onnistuminen.

4.2.2 ASP.NET Web API

ASP.NET Web API-sovelluskehys on tarkoitettu web-sovelluspalveluiden luontiin ja sen avulla voidaan luoda REST-arkkitehtuurityylin mukaisia web-sovelluspalveluita. Web

API hyödyntää Microsoftin .NET-sovelluskehystä ja sen ydin perustuu ASP.NET-teknologiaan. (Freeman 2014a, Ch. 4)

Web API:n suunnittelumalli on lähellä ASP.NET MVC-sovelluskehyksen hyödyntämää suunnittelumallia. ASP.NET Web API-sovelluskehyksen suunnittelumallin sisältämät osat ovat esillä alla olevassa kuvassa (Kuva 4.7). (Freeman 2014a, Ch. 4)



Kuva 4.7 ASP.NET Web API-suunnittelumalli (perustuu lähteeseen Freeman 2014a).

Suunnittelumalli koostuu ohjaimesta (Controller) ja mallista (Model). Erona ASP.NET MVC-suunnittelumalliin on näkymää kuvaavan osion puuttuminen. Kuten ASP.NET MVC-sovelluskehyksessäkin, mallin tehtävänä on säilyttää sovelluksen tarvitsema tietosisältö ja mallin luokkana voidaan käyttää mitä tahansa .NET-luokkaa. Ohjaimen tehtävänä on ottaa vastaan asiakkaan lähettämät HTTP-pyynnöt ja pyytää mallia tekemään niiden mukaiset muutokset. Ohjain on myös vastuussa mallin tietojen välittämisestä edelleen takaisin asiakkaalle.

Vaikka suunnittelumalli sisältää samoja osia kuin ASP.NET MVC, sijaitsevat Web API:n osien komponentit eri nimiavaruudessa ja ne sisältävät myös oman toteutuksensa, eivätkä täten ole suoraan yhteensopivia ASP.NET MVC:n komponenttien kanssa. Molempia sovelluskehyksiä voidaan kuitenkin käyttää rinnakkain ja ne voivat esimerkiksi jakaa samat malliluokat. (Freeman 2014a, Ch. 4; Ugurlu et al. 2013, Ch. 1)

Web API:n ohjain-luokat periytetään `ApiController`-kantaluokasta. Seuraavalla sivulla olevassa kuvassa (Kuva 4.8) on näkyvillä esimerkki Web API ohjain-luokan toteutuksesta. Esimerkissä on näkyvissä selvyyden vuoksi myös mallia kuvaavan `Product`-luokan tietojen alustus. Oikeassa projektissa tietojen alustus tehtäisiin erillisessä luokassa ja tiedot voitaisiin hakea esimerkiksi tietokannasta.

Sovelluskehys luo luokan julkisille metodeille automaattisesti URL-osoitteet, joita asiakassovellukset voivat kutsua HTTP-pyyntöjen avulla. `PersonsController`-luokan metodien URL-osoitteet ovat näkyvissä metodin yläpuolella olevissa kommenteissa. Ohjaimen julkiset metodit käyttävät hyväkseen REST-arkkitehtuurityylin mukaisesti HTTP-metodeja, joilla voidaan ilmaista onko kyseessä tiedon hakeminen, uuden tiedon luominen, tiedon päivittäminen tai poistaminen.

Tietoa haettaessa ohjaimen metodit palauttavat malleja kuvaavia olioita, jotka sovelluskehys kääntää automaattisesti JSON- tai XML-formaattiin. Asiakassovellus voi määritellä haluamansa formaatin HTTP-pyynnön otsikkotiedoissa `Content-Type`-kentän avulla.

Metodit palauttavat tämän lisäksi HTTP-vastauksen otsikkotiedoissa HTTP-statuskoodin, jonka avulla asiakassovellukset voivat päätellä operaation onnistumisen.

```
public class PersonsController : ApiController
{
    private static List<PersonModel> persons = new List<PersonModel>()
    {
        new PersonModel { Id = 1, UserName = "erkki123", Email = "erkki@example.com" },
        new PersonModel { Id = 2, UserName = "teemu", Email = "teemu@example.com" },
        new PersonModel { Id = 3, UserName = "ahto", Email = "ahto@example.com" }
    };

    // GET: Hae kaikki henkilöt, http://localhost/api/Persons/
    public IEnumerable<PersonModel> GetPersons()
    {
        return persons;
    }

    // GET: Hae henkilö id:n perusteella, http://localhost/api/Persons/2
    public IHttpActionResult GetPerson(int id)
    {
        var person = persons.FirstOrDefault((p) => p.Id == id);
        if (person == null) { return NotFound(); }
        return Ok(person);
    }

    // POST: Luo henkilö, http://localhost/api/Persons/
    public IHttpActionResult PostPerson(PersonModel value)
    {
        persons.Add(value);
        return Ok();
    }

    // PUT: Päivitä henkilön tiedot id:n perusteella, http://localhost/api/Persons/2
    public IHttpActionResult PutPerson(int id, PersonModel value)
    {
        var person = persons.FirstOrDefault((p) => p.Id == id);
        if (person == null) { return NotFound(); }
        person = value;
        return Ok();
    }

    // DELETE: Poista henkilö id:n perusteella, http://localhost/api/Persons/2
    public IHttpActionResult Delete(int id, PersonModel value)
    {
        var person = persons.FirstOrDefault((p) => p.Id == id);
        if (person == null) { return NotFound(); }
        persons.Remove(person);
        return Ok();
    }
}
```

Kuva 4.8 Esimerkki ASP.NET Web API ohjain-luokasta.

Alla olevassa taulukossa (Taulukko 4.2) on esitetty edellisessä esimerkissä kuvatun `PersonsController`-luokan `GetPersons`-metodin palauttama JSON- ja XML-formaatin mukainen vastaus. Esimerkistä on poistettu selvyyden vuoksi osa HTTP-vastauksen otsikkotiedoista.

Taulukko 4.2 PersonsController-luokan GetPersons-metodin palauttama JSON- ja XML-formaatin mukainen vastaus.

Http-vastaus	**JSON-formaatissa**

Otsikkotiedot	`HTTP/1.1 200 OK`
	`Content-Type: application/json; charset=utf-8`
	`Content-Length: 170`

Http-vastaus	`[{`
	`"Id" : 1,`
	`"UserName" : "erkki123",`

<table>
<tr><td></td><td><pre>
                "Email" : "erkki@example.com"
        }, {
                "Id" : 2,
                "UserName" : "teemu",
                "Email" : "teemu@example.com"
        }, {
                "Id" : 3,
                "UserName" : "ahto",
                "Email" : "ahto@example.com"
        }
]
</pre></td></tr>
<tr><td colspan="2">Http-vastaus XML-formaatissa</td></tr>
<tr><td>Otsikkotiedot</td><td><pre>
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Content-Length: 446
</pre></td></tr>
<tr><td>Http-vastaus</td><td><pre>
<ArrayOfPersonModel xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://schemas.datacontract.org/2004/07/WebAPI2Test.Models">
        <PersonModel>
                <Email>erkki@example.com</Email>
                <Id>1</Id>
                <UserName>erkki123</UserName>
        </PersonModel>
        <PersonModel>
                <Email>teemu@example.com</Email>
                <Id>2</Id>
                <UserName>teemu</UserName>
        </PersonModel>
        <PersonModel>
                <Email>ahto@example.com</Email>
                <Id>3</Id>
                <UserName>ahto</UserName>
        </PersonModel>
</ArrayOfPersonModel>
</pre></td></tr>
</table>

Taulukosta nähdään, että `GetPersons`-metodi palauttaa HTTP-vastauksen otsikkotiedoissa HTTP-statuskoodin 200, joka tarkoittaa HTTP-pyynnön onnistuneen. Otsikkotiedoissa kerrotaan myös vastauksena tulevan datan tyyppi `Content-Type`-kentässä ja vastausdatan koko tavuina `Content-Length`-kentän avulla. HTTP-vastaus sisältää varsinaisen datan JSON- tai XML-formaatissa. XML- ja JSON-formaatin etuna on se, että sen käsitteleminen on helpompaa ohjelmallisesti. Se voidaan esimerkiksi muuntaa suoraan olioiksi asiakassovelluksen muistiin sarjallistamisen avulla. Tällöin asiakassovelluksessa voidaan käyttää samanlaisia mallin tietosisältöä kuvaavia luokkia kuin palvelimellakin.

4.2.3 Kolmannen osapuolen kirjastot

Tässä kohdassa esitellään tietomallipalvelimen toteutuksessa käytettävät kolmannen osapuolen kirjastot. Ensimmäisenä esitellään tietomallipalvelimelle lähetettyjen IFC-tiedostojen parsintaan käytettävä IFC Engine-kirjasto. Seuraavaksi käydään läpi tietomallipalvelimella suoritettavien pitkäaikaisten tehtävien hallintaan käytettävä Hangfire-kirjasto. Kirjastoa käytetään apuna, kun suoritetaan tietomallipalvelimelle lähetettyjen IFC-tiedostojen tarkastaminen ja tallentaminen tietokantaan.

IFC Engine. IFC Engine on C++-ohjelmointikielellä toteutettu ohjelmointikirjasto IFC-tiedostojen käsittelyyn. Kirjaston kehittäminen on aloitettu vuonna 2001 ja nykyään se on käytössä monen kaupallisen sovelluksen taustalla. Kirjaston avulla IFC-tiedostoja voidaan lukea, muokata ja luoda. IFC Engine tarjoaa luokat ja metodit, joilla voidaan lukea

IFC-tiedoston sisältämä meta- ja geometriatieto. Kirjasto osaa käsitellä STEP- ja XML-formaatin mukaisia IFC-tiedostoja ja se sisältää tuen IFC-standardin versioille IFC 2x3 TC1 ja IFC4. Kirjastolle on toteutettu C#-ohjelmointikielellä wrapper-luokka, jonka avulla kirjastoa voi käyttää myös C#-sovelluksissa. Wrapper-luokan toteutus on esitetty liitteessä A. (IFC Engine)

Alla olevassa kuvassa (Kuva 4.9) on esimerkki IFC Engine kirjaston käytöstä C#-ohjelmointikielellä. Esimerkissä hyödynnetään kirjastolle toteutettua C# Wrapper-luokkaa. Esimerkissä avataan kirjaston avulla IFC2x3 TC1-version mukainen IFC-tiedosto ja haetaan kaikki tiedostosta löytyvät palkkia kuvaavat `IfcBeam`-objektit. Löydettyjen `IfcBeam`-objektien `GlobalId`- ja `Name`-attribuuttien arvot tulostetaan konsoliin.

```
public void openIfcFile(string filePath)
{
    // Avataan ifc-tiedosto
    long ifcModel = IFCEngineModule.IFCEngineWrapper.sdaiOpenModelBN(0, filePath, "IFC2X3_TC1.exp");
    // Tarkistetaan onnistuiko avaus
    if (ifcModel != 0)
    {
        string ifcEntityName = "IfcBeam";
        // Haetaan ifc-tiedostosta kaikki objektit joiden tyyppi on IfcBeam.
        long ifcObjectInstances = IFCEngineModule.IFCEngineWrapper.sdaiGetEntityExtentBN(ifcModel, ifcEntityName);
        long ifcObjectInstancesCount = IFCEngineModule.IFCEngineWrapper.sdaiGetMemberCount(ifcObjectInstances);

        if (ifcObjectInstancesCount != 0)
        {
            // Käydään läpi ifc-tiedostosta löytyneet IfcBeam-objektit.
            for (long i = 0; i < ifcObjectInstancesCount; ++i)
            {
                // Haetaan IfcBeam-objekti
                long ifcObjectIns = 0;
                IFCEngineModule.IFCEngineWrapper.engiGetAggrElement(ifcObjectInstances, i,
                                                                    IFCEngineModule.IFCEngineWrapper.sdaiINSTANCE,
                                                                    ref ifcObjectIns);

                // Haetaan IfcBeam-objektin GlobalId-attribuutin arvo.
                IntPtr value = IntPtr.Zero;
                IFCEngineModule.IFCEngineWrapper.sdaiGetAttrBN(ifcObjectIns, "GlobalId",
                                                               IFCEngineModule.IFCEngineWrapper.sdaiSTRING,
                                                               out value);
                string globalID = Marshal.PtrToStringAnsi((IntPtr)value);

                // Haetaan IfcBeam-objektin Name-attribuutin arvo.
                value = IntPtr.Zero;
                IFCEngineModule.IFCEngineWrapper.sdaiGetAttrBN(ifcObjectIns, "Name",
                                                               IFCEngineModule.IFCEngineWrapper.sdaiSTRING,
                                                               out value);
                string name = Marshal.PtrToStringAnsi((IntPtr)value);

                // Tulostetaan IfcBeam-objektin GlobalId- ja Name-attribuuttien arvot konsoliin.
                System.Console.WriteLine("Global id: " + globalID + ", Name: " + name);
            }
        }
    }
}
```

Kuva 4.9 Esimerkki IFC Engine-kirjaston käytöstä C#-ohjelmointikielellä.

Hangfire. Hangfire on C#-ohjelmointikielellä toteutettu avoimen lähdekoodin kirjasto, jonka avulla voidaan suorittaa erilaisia pitkäaikaisia tehtäviä ASP.NET-sovelluksissa (Hangfire). Kirjaston avulla pitkään kestävät operaatiot voidaan suorittaa web-sovelluksen taustaprosessina asynkronisesti. Tällöin palvelin voi käsitellä tehtävän suorituksen aikana myös muita käyttäjän lähettämiä HTTP-pyyntöjä, eikä käyttäjän ei tarvitse jäädä odottamaan tehtävän suoritusta. (Hangfire overview)

Kirjaston avulla voidaan suorittaa kolmen tyyppisiä tehtäviä. Ensimmäisenä ovat kerran suoritettavat operaatiot. Toisena ovat viivästetyt tehtävät, jotka voidaan suorittaa annetun aikarajan jälkeen. Kolmantena ovat toistuvat tehtävät, joita voidaan suorittaa esimerkiksi päivittäin. Alla olevassa kuvassa (Kuva 4.10) on esimerkit eri tyyppisten tehtävien suorittamisesta.

```
public void doHangFireTasks()
{
    // Kerran suoritettava tehtävä
    BackgroundJob.Enqueue(() => Console.WriteLine("Hei maailma!"));

    // Viivästetty tehtävä
    BackgroundJob.Schedule(() => Console.WriteLine("Hei maailma!"), TimeSpan.FromDays(1));

    // Toistuva tehtävä
    RecurringJob.AddOrUpdate(() => Console.Write("Hei maailma!"), Cron.Daily);
}
```

Kuva 4.10 Esimerkki Hangfire-kirjaston käytöstä.

Esimerkistä nähdään, että kerran suoritettavien tehtävien suorittamiseen voidaan käyttää Hangfire-kirjaston `BackgroundJob`-luokan `Enqueue`-metodia. Metodille voidaan antaa parametrina suoritettava funktio. Viivästettyjen tehtävien suorittamiseen `BackgroundJob`-luokka tarjoaa `Schedule`-metodin, jolle voi antaa suoritettavan funktion lisäksi aikamääreen, jolloin tehtävä suoritetaan. Toistuvien tehtävien suorittamista varten käytetään `RecurringJob`-luokkaa. Luokka tarjoaa `AddOrUpdate`-metodin, jolle voidaan määritellä suoritettava funktio ja suorituksen toistuvuus.

Kirjasto käyttää apunaan tietokantaa, johon annetut tehtävät tallennetaan. Tällä voidaan varmistua siitä, että tehtävä tulee suoritetuksi vaikka palvelin uudelleenkäynnistettäisiin. Tietokantaa käytetään apuna myös niissä tilanteissa, joissa tehtävän suoritus ei onnistu ensimmäisellä kerralla. Hangfire-kirjasto osaa yrittää tehtävän suoritusta uudelleen tietyn aikarajan jälkeen. Tällä hetkellä tuki löytyy Microsoft SQL-, redis, PostgreSQL-, MongoDB- ja composite-tietokannoille. (Hangfire overview)

Tehtävien hallintaa varten Hangfire tarjoaa selainpohjaisen tehtävien monitorointinäkymän. Näkymän avulla jonossa olevia ja jo suoritettuja tehtäviä voi tarkastella ja hallita. Näkymää voidaan käyttää myös epäonnistuneiden tehtävien poistamiseen jonosta. (Hangfire overview)

4.2.4 Kehitysympäristö ja ohjelmointikieli

Tietomallipalvelimen kehittämiseen käytetään Microsoft Visual Studio 2013 Professional-kehitysympäristöä (Visual Studio). Visual Studio on Microsoftin kehittämä graafinen sovellusten kehitysympäristö, jossa voi voi hyödyntää useita eri ohjelmointikieliä, kuten C++, C#, Visual Basic ja F# (Visual Studio languages). Kehitysympäristö valittiin, koska se pitää sisällään valmiit projektipohjat ASP.NET MVC- ja ASP.NET Web API-

sovelluksille. Visual Studio ymmärtää myös näkymän toteutuksessa käytettävää Razor-merkkauskieltä ja osaa ilmoittaa sen syntaksivirheistä. Kehitysympäristö pitää sisällään myös IIS-palvelimen, jonka avulla kyseisiä web-sovelluksia on mahdollista suorittaa ja debugata paikallisesti. Samaa kehitysympäristöä käytetään myös tietomallipalvelimen varmuuskopioiden palautuksen suorittavan sovelluksen luomiseen.

Tietomallipalvelimen ohjelmointiin käytetään C#-ohjelmointikieltä. C# on Microsoftilla kehitetty oliopohjainen ohjelmointikieli, jonka ensimmäinen versio julkaistiin heinäkuussa 2000 osana Microsoftin .NET-sovelluskehystä. C# on syntaksiltaan ja rakenteiltaan lähellä C, C++ ja Java-ohjelmointikieliä. C# on vahvasti tyypitetty ja sen tietotyypit ovat `reference-` tai `value-`tyyppisiä. `Reference`-tyyppien kantaluokkana toimii `Object`-luokka, josta muut erilaisia tietotyyppejä kuvaavat luokat periytetään. Value-tyypit ovat perustietotyyppejä kuten `bool`, `int`, `float` ja `char`, joilla kuvataan totuusarvoja, kokonaislukuja, liukulukuja ja merkkijonoja. C# sisältää myös funktionaalisten ohjelmointikielten piirteitä kuten lambda-lausekkeet joiden avulla voidaan esittää nimettömiä funktoita ja LINQ-kyselykielen, jonka avulla voidaan suorittaa kyselyitä esimerkiksi SQL-tietokantaan tai XML-dokumenttiin. C# sisältää myös automaattisen muistinhallinnan, joka huolehtii esimerkiksi sovelluksen varaaman muistin automaattisesta vapauttamisesta. C#-ohjelmointikielen määrittelyt löytyvät standardeista ISO/IEC 23270 ja ECMA-334. (ISO/IEC 23270; ECMA-334; Introduction to LINQ Queries)

Ohjelmointikieli valittiin, koska useimmat ASP.NET-MVC- ja ASP.NET Web API-sovelluskehykselle kirjoitetut koodiesimerkit ja lähdemateriaali on kirjoitettu kyseisellä ohjelmointikielellä. Samaa ohjelmointikieltä käytetään myös Tekla Open API-laajennoksen luomiseen.

4.3 Käyttöliittymä

Tietomallipalvelimen käyttöliittymä esitetään selaimessa. Käyttöliittymän toteutuksessa käytetään apuna ASP.NET MVC-sovelluskehyksen tarjoamia näkymämoottoreita, joilla tuotetaan CSS-tyylien avulla muotoiltuja HTML-sivuja. Sivujen muodostuksessa käytetään apuna jQuery JavaScript-kirjastoa, jonka avulla sivujen sisältöä voidaan muokata dynaamisesti. Tietomallipalvelimen tietomallien tarkastelunäkymän toteutuksessa käytetään tämän lisäksi WebGL-teknologiaa ja Three.js-kirjastoa. Näistä teknologioista on kerrottu tarkemmin seuraavissa aliluvuissa.

4.3.1 jQuery

Käyttöliittymän toteutuksessa hyödynnettävä jQuery on avoimen lähdekoodin JavaScript-kirjasto. Kirjaston tarkoituksena on yksinkertaistaa ja nopeuttaa sovelluskehitystä. Se tarjoaa monia valmiita metodeja, joiden toteuttaminen pelkästään JavaScriptin avulla vaatisi useampia rivejä koodia. Kirjasto julkaistiin vuonna 2006 ja sen kehittäjä on John Resig, mutta tällä hetkellä kehityksestä vastaa jQuery-tiimi. Kirjasto on myös osa

ASP.NET MVC-sovelluskehystä, jossa sitä käytetään apuna HTML-sivujen dynaamisessa muokkauksessa. (McGrath 2014, Ch. 1)

jQuery on yhteensopiva useimpien selaimien kanssa, joten se auttaa myös selainten vaihtelevan JavaScript-tuen hallinnassa. Selainten käyttämät JavaScript-moottorit eroavat hieman toisistaan, joten sama JavaScript-koodi ei välttämättä toimi kaikissa selaimissa samalla tavalla. Koska kirjasto abstrahoi JavaScript-koodin tarjoamiensa metodien taakse, voi sovelluskehittäjä tarjota niitä käyttämällä tuen useimmille selaimille ilman erillistä optimointia.

Käytännössä kirjasto luo HTML-dokumentin rakennetta ja sen elementtejä kuvaavan DOM-puun (Document Object Model). DOM-puun sisältämät elementit ovat jQuery-objekteja, jotka tarjoavat erilaisia funktioita. Kirjasto tarjoaa toiminnot mm. HTML-dokumentin sisältämien elementtien ja CSS-tyylien muokkaukseen. Tämän lisäksi se mahdollistaa reagoimisen käyttäjän toimintoihin tapahtumankäsittelijöiden avulla sekä erilaisten animaatioiden luomisen. Kirjasto sisältää myös yksinkertaistetut metodit AJAX-kutsujen luontiin ja niiden vastausten käsittelyyn. (McGrath 2014, Ch. 1)

Kirjaston päälle on myös mahdollista kehittää omia liitännäisiä, joiden avulla voi toteuttaa esimerkiksi omia käyttöliittymäkomponentteja tai animaatioita. Yksi tunnetuimmista liitännäisistä on JQuery UI-kirjasto (https://jqueryui.com/), joka on tarkoitettu web-sovellusten käyttöliittymien rakentamiseen.

jQueryn komentojen syntaksi on seuraava:

```
$(valitsija).funktio()
```

Jokainen komento pitää sisällään tavallisesti valitsijan, jonka avulla voidaan valita yksi tai useampi HTML-dokumentin elementti. Valitsijana käytetään `$()`-funktiota. Funktion parametriksi annetaan elementin hakuehto. Hakuehtoina voidaan käyttää HTML-elementin tyyppiä, sekä elementin class- tai id-attribuutin arvoa. Funktio palauttaa jQuery-objekteja, jotka ovat HTML-dokumentista löytyneitä kyseistä hakuehtoa vastaavia elementtejä. Palautetuille objekteille voidaan suorittaa tämän jälkeen haluttu toiminto. Palautetulle objekteille voidaan asettaa myös tapahtumankäsittelijä, jota kutsutaan esimerkiksi kun elementtiä klikataan hiirellä. (McGrath 2014, Ch. 1, 6)

Tämän lisäksi jQuery pitää sisällään myös `$.`-notaatiolla alkavia apumetodeja. Apumetodien avulla voidaan lähettää esimerkiksi HTTP-pyyntöjä tai AJAX-kutsuja. (McGrath 2014, Ch. 1, 6)

Seuraavan sivun kuvassa (Kuva 4.11) on esimerkki jQuery-kirjaston käytöstä. Esimerkissä asetetaan `button`-elementin klikkaukselle tapahtumankäsittelijä. Kun elementtiä klikataan, asetetaan `input`-kentän sisältämä syöte `p`-elementin sisällöksi.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>jQuery testi</title>
        <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
        <script>
            // Funktiota ready kutsutaan kun kaikki HTML-dokumentin elementit on ladattu.
            $(document).ready(function () {
                // Kutsutaan button-elementtiä klikattaessa.
                $("#nappi").click(function () {
                    // Haetaan input-kentän arvo
                    var inputText = $("#syote").val();
                    // Asetetaan input-kentän arvo p-elementin sisällöksi.
                    $("#tulos").text(inputText);
                });
            });
        </script>
    </head>
    <body>
        <input type="text" id="syote"></input>
        <button id="nappi">Aseta arvo</button>
        <p id="tulos"></p>
    </body>
</html>
```

Kuva 4.11 Esimerkki jQuery-kirjaston käytöstä.

Esimerkissä käytetään valitsijoina elementtien `id`-attribuuttien arvoja. Tämän lisäksi esimerkissä on hyödynnetty jQuery-kirjaston tarjoamaa `val`-funktiota, jolla voidaan hakea HTML-elementin sisältämä arvo, sekä `text`-funktiota, jolla voidaan asettaa HTML-elementin sisältö. Esimerkin toteuttava koodi on sijoitettu `ready`-funktion sisään. Funktiota kutsutaan, kun koko HTML-dokumentti on ladattu. Tällä voidaan varmistua siitä, että kaikki HTML-dokumentin sisältämät elementit ovat ladattu ja käytettävissä.

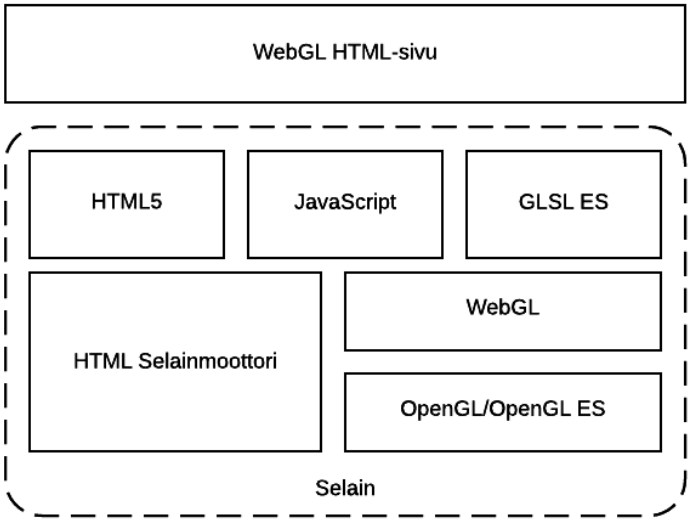
4.3.2 WebGL

WebGL eli Web Graphics Library on teknologia, jonka avulla voidaan esittää 3D-grafiikkaa selaimessa. WebGL mahdollistaa näytönohjaimen laitteistokiihdytyksen hyödyntämisen selaimessa ilman erillisten laajennosten asentamista. WebGL vaatii toimiakseen tuen selaimelta, mutta useimmat nykyselaimista, kuten Apple Safari (http://www.apple.com/fi/safari/), Google Chrome (https://www.google.com/chrome), Mozilla Firefox (https://www.mozilla.org/fi/firefox/new/) ja Opera (http://www.opera.com/fi) sisältävät tuen WebGL:lle. (WebGL Specification 2014; WebGL Public Wiki 2015)

WebGL:n juuret ovat Silicon Graphicsin vuonna 1992 julkaisemassa avoimessa OpenGL-standardissa (Open GL), joka on laitteistoriippumaton ohjelmointirajapinta 2D- ja 3D-grafiikan esittämiseen pöytätietokoneissa. OpenGL:stä kehitettiin vuosina 2003-2004 sulautetuille laitteille tarkoitettu OpenGL ES (OpenGL ES), joka sisältää hieman vähemmän ominaisuuksia kuin OpenGL. Vuonna 2007 julkaistiin OpenGL ES:n versio 2.0, johon WebGL perustuu. WebGL:n kehitys aloitettiin vuonna 2009 ja sen ensimmäi-

nen versio 1.0 julkaistiin vuonna 2011. Kehittämisestä vastaa Khronos Group, johon kuuluu monia selainvalmistajia, kuten Apple, Google, Microsoft, Mozilla ja Opera. (Danchilla 2010, WebGL Origins; WebGL Public Wiki 2015)

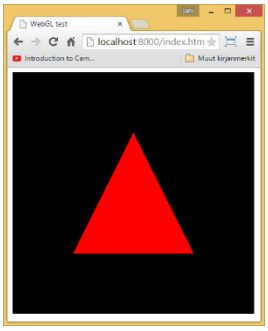
WebGL:ää hyödyntävään HTML-sivuun liittyvät teknologiat on esitelty alla olevassa kuvassa (Kuva 4.12). Käytännössä WebGL on JavaScriptin avulla käytettävä ohjelmointirajapinta, jonka kautta voidaan välittää piirtokäskyjä suoraan näytönohjaimelle.



Kuva 4.12 WebGL-arkkitehtuuri (perustuu lähteeseen Lea & Matsuda 2013, Figure 1.5).

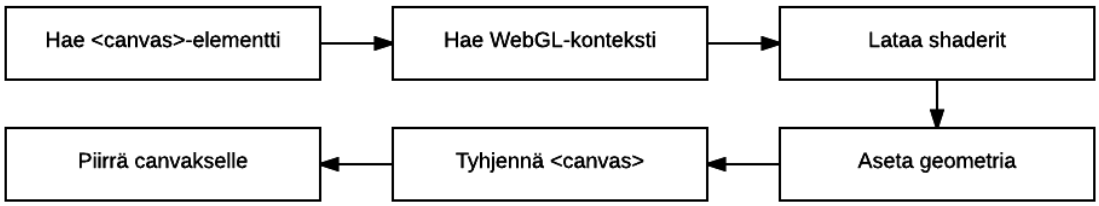
WebGL käyttää hyväkseen HTML5-teknologiaa ja sen tarjoamaa `canvas`-elementtiä grafiikan esittämisessä. Grafiikan piirtämisessä käytetään apuna shadereita. Shaderit kuvataan OpenGL ES shading language avulla, josta käytetään myös nimitystä GLSL ES (The OpenGL ES Shading Language). Shadereiden avulla määritellään esimerkiksi piirrettävä geometria ja sen väritys.

Alla olevassa kuvassa (Kuva 4.13) on esimerkki WebGL:n avulla piirretystä kolmiosta Chrome-selaimessa. Sovelluksen lähdekoodi löytyy kokonaisuudessaan liitteestä B.



Kuva 4.13 WebGL:n avulla piirretty kolmio Chrome-selaimessa.

Alla olevassa kuvassa (Kuva 4.14) on esitetty yksinkertaistetusti Liitteestä B löytyvän WebGL-sovelluksen suorituksen eteneminen vuokaavion avulla. WebGL-sovelluksen logiikka määritellään JavaScriptin avulla. Sovelluksen JavaScript-koodi on sijoitettu suoraan HTML-dokumentin sisälle, mutta parempi käytäntö olisi määritellä koodi erillisessä tiedostossa, johon viitataan HTML-dokumentin `script`-elementin avulla.



Kuva 4.14 WebGL-sovelluksen suorituksen eteneminen.

Sovelluksen suoritus aloitetaan hakemalla JavaScriptin avulla piirtoon käytettävä `canvas`-elementti HTML-dokumentin sisältä. Seuraavaksi pyydetään selaimelta WebGL-konteksti `canvas`-elementin tarjoaman `getContext`-metodin avulla. Alla olevassa kuvassa (Kuva 4.15) on canvas-elementin ja WebGL-kontekstin hakemiseen käytetty JavaScript-koodi. WebGL-konteksti voi olla määritetty, joko ”webgl”- tai ”exiperimental-webgl”-määreellä. Jälkimmäisellä pyritään ilmaisemaan se, että kyseinen selain ei vielä toteuta kaikkia WebGL:n ominaisuuksia. WebGL-konteksti tarjoaa metodit mm. piirtokoodin välittämiselle näytönohjaimelle sekä `canvas`-elementin käsittelyyn ja siihen piirtämiseen (Lea & Matsuda 2013, p. 19-20).

```
// Haetaan canvas-elementti id:n perusteella HTML-dokumentista,
// johon piirtäminen suoritetaan.
var canvas = document.getElementById("webgl");
var gl = null;

try {
    // Haetaan webgl-konteksti gl-muuttujaan.
    gl = canvas.getContext("webgl") || canvas.getContext("experimental-webgl");
}
catch (e) { }
```

Kuva 4.15 Canvas-elementin ja WebGL-kontekstin hakeminen JavaScriptin avulla.

Seuraavaksi sovelluksessa haetaan shadereiden määrittelyyn käytetty koodi. WebGL:ssä käytetään kahdentyyppisiä shadereita, joita ovat vertex- ja fragment-shaderit. Vertex-shadereiden avulla määritellään pistejoukon avulla piirrettävä geometria ja yksittäisten pisteiden väritys. Fragment-shadereita käytetään määrittelemään lopullinen väritys, jossa otetaan huomioon esimerkiksi valaistus. Fragment-shadereiden avulla on mahdollista tuottaa myös erilaisia efektejä, kuten sumua. Shadereille voidaan määrittää attribuutteja, jotka ovat käsiteltävissä myös JavaScript-koodissa. Tällä tavoin voidaan muokata esimerkiksi esitettävää geometriaa sovelluksen suorituksen aikana. (Danchilla 2010, p. 36-37; Lea & Matsuda 2013, p. 41)

Alla olevassa kuvassa (Kuva 4.16) on vertex- ja fragment-shaderit määrittelevä koodi. GLSL ES-koodi on sijoitettu HTML-sivulle `script`-elementtien sisälle. Vertex-shaderille voi antaa attribuuttina piirrettävän pisteen koordinaatin. Fragment-shaderissa asetetaan piirrettävän pisteen väri punaiseksi.

```
<!-- Vertex-shader -->
<script id="shader-vs" type="x-shader/x-vertex">
    attribute vec3 aVertexPosition;
    void main() {
        gl_Position = vec4(aVertexPosition, 1.0);
    }
</script>

<!-- Fragment-shader -->
<script id="shader-fs" type="x-shader/x-fragment">
    void main() {
        gl_FragColor = vec4(1.0,0.0,0.0, 1.0);
    }
</script>
```

Kuva 4.16 WebGL vertex- ja fragment-shaderin määrittelyt.

GLSL ES-koodi haetaan HTML-dokumentista JavaScriptin avulla. Tämän jälkeen GLSL ES-koodi käännetään ja linkitetään osaksi sovellusta WebGL-kontekstin tarjoamien metodien avulla. Alla olevassa kuvassa (Kuva 4.17) on koodi, jossa suoritetaan shader-koodin alustaminen ja lataus. Koodissa hyödynnetään aiemmin määriteltyjä vertex- ja fragment-shadereita.

```
// Luodaan shader-oliot
var vertexShader = gl.createShader(gl.VERTEX_SHADER);
var fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);

// Haetaan GLSL ES-koodi HTML-dokumentin script-elementeiltä id:n perusteella
var vertexShaderSource = document.getElementById('shader-vs').innerHTML;
var fragmentShaderSource = document.getElementById('shader-fs').innerHTML;

// Asetetaan shadereiden lähdekoodi
gl.shaderSource(vertexShader, vertexShaderSource);
gl.shaderSource(fragmentShader, fragmentShaderSource);

// Käännetään shadereiden lähdekoodi
gl.compileShader(vertexShader);
gl.compileShader(fragmentShader);

// Luodaan sovellus
var glProgram = gl.createProgram();
// Liitetään shaderit sovellukseen
gl.attachShader(glProgram, vertexShader);
gl.attachShader(glProgram, fragmentShader);
// Linkataan sovellus
gl.linkProgram(glProgram);

// Tarkistetaan onnistuko linkkaus
if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
    // Virhe
}

// Käsketään webgl-kontekstin käyttää sovellusta.
gl.useProgram(glProgram);
```

Kuva 4.17 Esimerkki WebGL shader-koodin hakemisesta, kääntämisesta ja linkittämisestä osaksi sovellusta.

Shader-koodin alustuksen jälkeen, sovelluksessa asetetaan piirrettävä geometria. Alla olevassa kuvassa (Kuva 4.18) on koodi, jolla kolmiota kuvaava geometria alustetaan. Geometria määritellään `vertices`-taulukon avulla, joka sisältää geometriaa kuvaavat xyz-koordinaatistossa sijaitsevat pisteet. Määrittelyn jälkeen taulukko ladataan välimuistiin, josta se on myöhemmin käytettävissä WebGL-kontekstin tarjoamien metodien avulla.

```
// Määritetään geometrian pisteet (x,y,z)-koordinaateilla.
var vertices = [
    -0.5, -0.5, 0.0,
    0.0, 0.5, 0.0,
    0.5, -0.5, 0.0
];

// Luodaan bufferi, johon geometrian pisteet asetetaan.
var verticeBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, verticeBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
```

Kuva 4.18 Esimerkki geometrian määrittelystä ja sen lataamisesta bufferiin.

Geometrian asetuksen jälkeen sovelluksessa asetetaan `canvas`-elementin tyhjentävä väri ja tyhjennetään `canvas`-elementti piirtoa varten. Lopuksi kutsutaan WebGL-kontekstin tarjoamia metodeja piirtämiselle. Alla olevassa kuvassa (Kuva 4.19) on koodi, jolla canvas-elementin tyhjentäminen ja piirtäminen suoritetaan. Koodissa hyödynnetään aiemmin määriteltyä vertex-shaderia ja geometriaa.

```
// Asetetaan tyhjä väri
gl.clearColor(0.0, 0.0, 0.0, 1.0);
// Tyhjennetään canvas
gl.clear(gl.COLOR_BUFFER_BIT);

// Sidotaan vertex-shader aVertexPosition-attribuutti geometrian määrittelevään bufferiin.
var vertexPositionAttribute = gl.getAttribLocation(glProgram, "aVertexPosition");
gl.enableVertexAttribArray(vertexPositionAttribute);
gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);

// Piirretään geometria canvas-elementille
gl.drawArrays(gl.TRIANGLES, 0, 3);
```

Kuva 4.19 Esimerkki canvas-elementin tyhjentämisestä ja siihen piirtämisestä WebGL-kontekstin avulla.

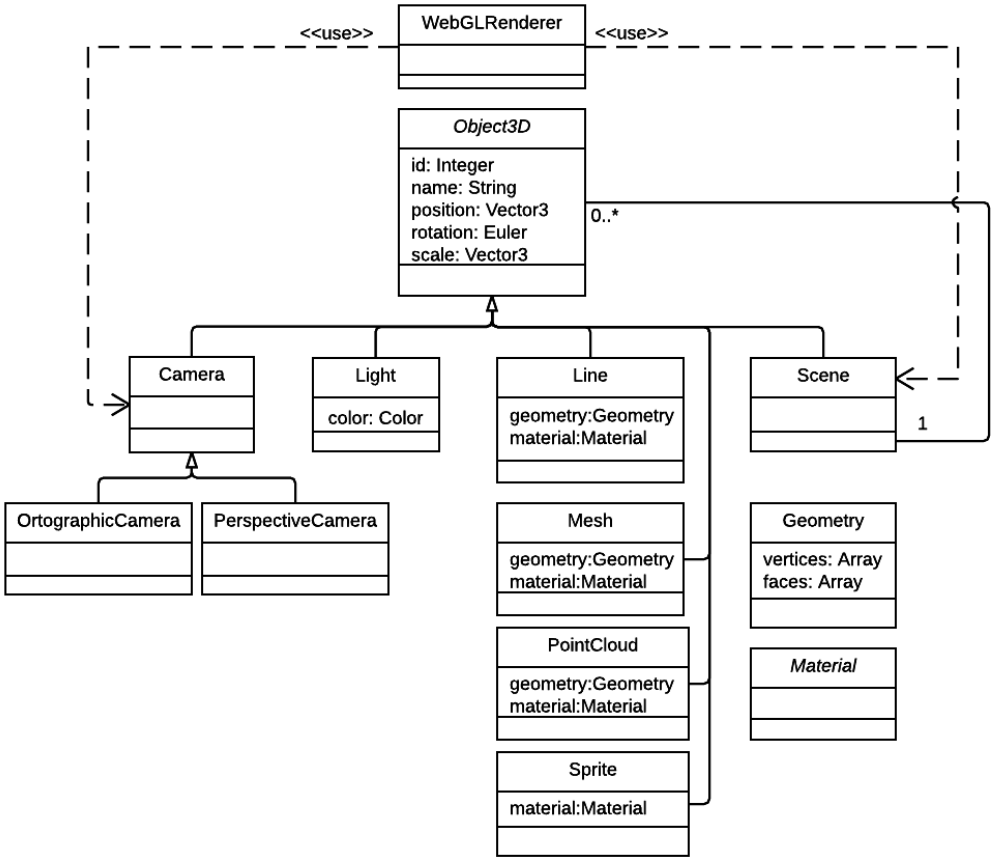
Piirtokutsun jälkeen näytönohjain suorittaa grafiikan laskennan ja piirtämisen näytönohjaimen muistissa. Kun grafiikka on saatu muodostettua, se piirretään selaimen `canvas`-elementille.

Esimerkistä voidaan havaita, että WebGL-teknologian hyödyntäminen vaatii useiden ohjelmointikielien hallintaa. Tämän takia sen käyttöönotto voi olla haastavaa, jos ei ole aiempaa kokemusta esimerkiksi OpenGL-ohjelmoinnista. Tästä syystä onkin kehitetty useita WebGL:n toimintaa abstrahoivia JavaScript-kirjastoja, joilla sovelluskehitystä voi nopeuttaa ja tehostaa.

4.3.3 Three.js

Tietomallipalvelimen tietomallin geometrian tarkasteluun tarkoitetun 3D-näkymän luontiin käytetään Three.js-kirjastoa (Three.js). Three.js on avoimen lähdekoodin Javascript-kirjasto 3D-grafiikan piirtoon selaimessa. Kirjaston kehitys aloitettiin vuonna 2010 Ricardo Cabellon toimesta, mutta nykyään sen kehittämisestä vastaa laaja yhteisö. (Danchilla 2012, Ch. 7; Three.js docs)

Three.js käyttää hyväkseen WebGL-teknologiaa. Kirjasto abstrahoi WebGL-rajapinnan ja tarjoaa monia peruskomponentteja, joiden avulla 3D-näkymän voi luoda. Alla olevassa kuvassa (Kuva 4.20) on luokkakaavio, jossa ovat näkyvillä Three.js-kirjaston tarjoamat tärkeimmät luokat ja niiden suhteet toisiinsa. Selvyyden vuoksi luokat sisältävät vain tärkeimmät attribuutit. Monet luokkakaaviossa näkyvistä luokista ovat abstrakteja kantaluokkia, joista varsinaiset luokkien ilmentymät on periytetty.



Kuva 4.20 Three.JS-kirjaston tärkeimmät luokat (Three.js docs).

Näkymän piirtäminen tehdään `WebGLRenderer`-luokan avulla. Luokka käyttää piirtämisessä hyväkseen kameraa kuvaavia `OrtoGraphicCamera-` tai `PerspectiveCamera`-luokkia sekä näkymää kuvaavaa `scene`-luokkaa. Three.js-kirjasto sisältää myös `CanvasRenderer`-luokan, jota voidaan hyödyntää niissä tapauksissa kun käytössä oleva näytönohjain ei tue WebGL:ää. (Three.js docs)

Kaikkien 3D-näkymässä olevien objektien kantaluokkana on `Object3D`. Luokka pitää sisällään attribuuttimäärittelyt `id` ja `name` joiden avulla näkymässä olevat objektit voidaan identifioida. `Position-`, `rotation-` ja `scale`-attribuuttien avulla voidaan määritellä ja muuttaa näkymässä olevan objektin sijaintia, rotaatiota ja kokoa. (Three.js docs, Object3D)

Three.js-kirjasto tarjoaa kaksi erilaista kameraa näkymän kuvaamiseen. Perspektiivin huomioonottavan 3D-näkymän esittämiseen käytetään `PerspectiveCamera`-luokan kameraa. `OrtographicCamera`-luokan kameraa voi hyödyntää niissä tilanteissa, joissa 3D-näkymä halutaan esittää 2D-projektiona.

Näkymän valaisemiseen käytetään `Light`-luokasta periytettyjä luokkia. Luokasta on periytetty esimerkiksi pistevalaistusta kuvaava `PointLight`-luokka, kohdevalaistusta kuvaava `SpotLight`-luokka ja yleisvalaistusta kuvaava `AmbientLight`-luokka, jonka avulla koko näkymä voidaan valaista tasaisesti. (Three.js docs)

Erilaisia geometrisia muotoja kuvataan luokkien `Line`, `Mesh` ja `PointCloud` avulla. `Line`-luokan avulla voidaan kuvata viivoista koostuvia geometrioita ja `PointCloud`-luokan avulla pisteistä koostuvia geometrioita. `Mesh`-luokkaa käytetään monimutkaisempien geometrioiden esittämiseen. (Three.js docs)

Geometriaa kuvaavat luokat käyttävät hyväkseen `Geometry`-luokkaa. Luokka sisältää attribuutit `vertices-` ja `faces`, joiden avulla geometriaa kuvaava kolmioverkko määritellään. `Vertices`-attribuutti pitää sisällään geometrian pisteet ja `faces`-attribuutti pisteiden järjestyksen, joka on määritetty kolmioiden avulla. (Three.js docs)

Objektien materiaalien määrittämiseen käytetään `Material`-luokasta periytettyjä luokkia. Materiaalia kuvaavia luokkia on erilaisia ja niiden avulla objektille voidaan määrittää mm. väri, läpinäkyvyys ja heijastuvuus. Materiaaleina voidaan myös käyttää kuvia. Tätä hyödyntää myös `Sprite`-luokka, jota käytetään 2D-kuvien näyttämiseen. (Three.js docs)

Näkymässä olevien objektien säilömiseen käytetään `Scene`-luokkaa. Luokan sisälle tulee lisätä kaikki ne objektit, joiden halutaan näkyvän näkymässä. Näitä ovat esimerkiksi geometriaa sisältävät objektit ja valaistus. Poikkeuksena tästä on näkymän kuvaamiseen käytettävä kamera. (Three.js docs)

Seuraavalla sivulla olevassa kuvassa (Kuva 4.21) on näkyvissä koodiesimerkki, jossa luodaan Three.js-kirjaston avulla 3D-näkymä, joka pitää sisällään punaisen kuution. Esimerkistä nähdään, että Three.js-kirjasto otetaan käyttöön viittaamalla siihen `script`-elementin avulla. 3D-näkymän piirtämisen toteuttava JavaScript-koodi on sijoitettu esimerkissä toisen `script`-elementin sisälle.

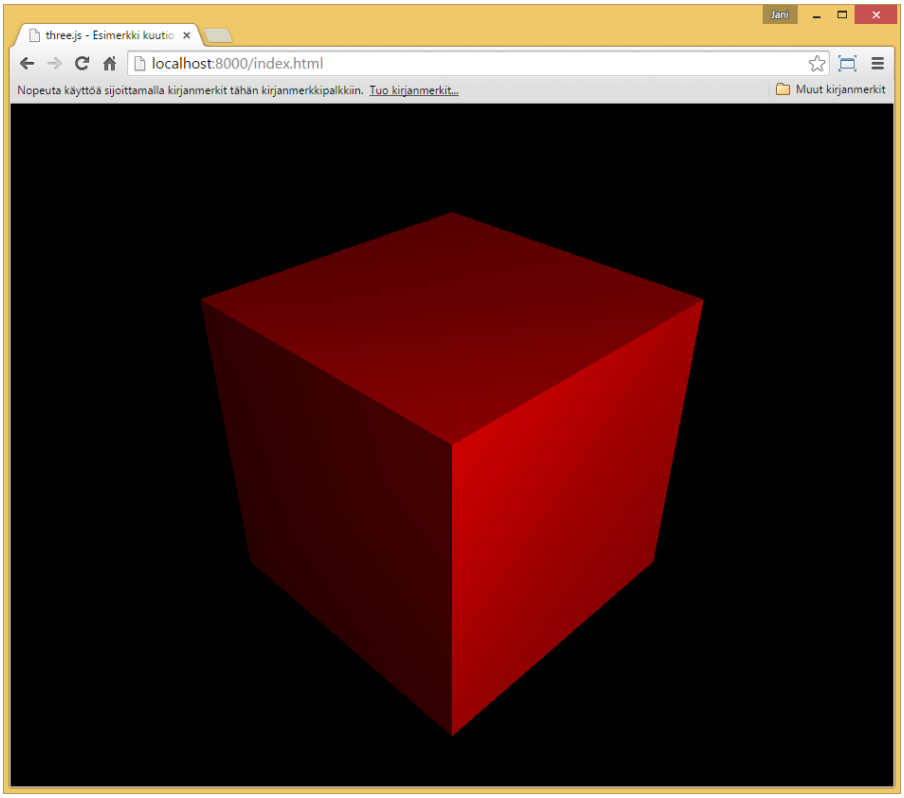
3D-näkymän alustus tehdään esimerkin `init`-funktiossa. Funktiossa luodaan aluksi näkymää kuvaava `Scene`-luokan olio. Tämän jälkeen luodaan kamera `PerspectiveCamera`-luokan avulla. Seuraavaksi luodaan `WebGLRenderer`-olio, joka huolehtii näkymän piirtämisestä. Tämän jälkeen lisätään JavaScriptin avulla HTML-dokumentin `body`-elementille `canvas`-elementti, jota `WebGLRenderer` käyttää näkymän piirtämiseen. Seuraavaksi luodaan kuution geometriaa ja materiaalia kuvaavat oliot Three.js-kirjaston tarjoamien `CubeGeometry`- ja `MeshLambertMaterial`-luokkien avulla. Kuutio luodaan `Mesh`-luokan avulla, jolle annetaan rakentajan parametrina kuvattavan objektin geometria ja materiaali. Kuutio lisätään näkymään `scene`-olion tarjoaman `add`-metodin avulla. Viimeisenä `init`-funktiossa luodaan näkymän valaistus, johon on käytetty `PointLight`-luokan oliota.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>three.js - Esimerkki kuution luonnista</title>
        <meta charset="utf-8">
        <style>
            body { margin: 0; overflow: hidden; } /* Poistetaan marginaalit ja viertyspalkki */
            canvas { width: 100%; height: 100% } /* Asetetaan canvas-elementin koko */
        </style>
    </head>
    <body>
        <script src="http://threejs.org/build/three.min.js"></script>
        <script>
            var camera, scene, renderer;
            init();
            animate();

            // Näkymän alustus-funktio
            function init() {
                // Luodaan scene- ja kamera-oliot.
                scene = new THREE.Scene();
                camera = new THREE.PerspectiveCamera( 45, window.innerWidth / window.innerHeight, 1, 1000 );
                camera.position.set( 15, 15, 15 );
                camera.lookAt( scene.position );

                // Luodaan renderöijä
                renderer = new THREE.WebGLRenderer();
                renderer.setSize( window.innerWidth, window.innerHeight );
                document.body.appendChild( renderer.domElement );

                // Luodaan kuution geometria ja materiaali.
                var cubeGeometry = new THREE.CubeGeometry( 10, 10, 10 );
                var cubeMaterial = new THREE.MeshLambertMaterial( { color: 0xff0000 } );

                // Luodaan kuutio ja lisätään se scene-oliolle.
                var cube = new THREE.Mesh( cubeGeometry, cubeMaterial );
                scene.add( cube );

                // Luodaan valaistus ja lisätään se scene-oliolle.
                var pointLight = new THREE.PointLight( 0xFFFF00 );
                pointLight.position.set( 20, 15, 10 );
                scene.add( pointLight );
            }

            // Näkymän piirtämis-funktio
            function animate() {
                requestAnimationFrame( animate );
                renderer.render( scene, camera );
            }
        </script>
    </body>
</html>
```

Kuva 4.21 Esimerkki kuution luomisesta Three.JS-kirjaston avulla.

Näkymän piirtäminen tapahtuu `animate`-funktiossa. Funktiossa kutsutaan ensin metodia `requestAnimationFrame`, jonka avulla renderöijä saa luvan piirtää näkymään. Tämän jälkeen kutsutaan metodia `render`, joka suorittaa näkymän piirtämisen. Metodille annetaan parametrina piirrettävä näkymää kuvaava `scene`- ja kameraa kuvaava `camera`-olio.

Alla olevassa kuvassa (Kuva 4.22) on näkyvissä koodiesimerkin luoma 3D-näkymä Chrome-selaimessa. Kuvasta nähdään, että 3D-näkymä skaalautuu selaimen ikkunan kokoiseksi. `Canvas`-elementin kokoa on säädetty esimerkissä CSS-tyylien avulla. Tyylimäärittelyt on sijoitettu esimerkissä HTML-dokumentin `style`-elementin sisälle ja niillä asetetaan `canvas`-elementin koko skaalautumaan selaimen ikkunan kokoon.



Kuva 4.22 Three.js-kirjastolla luotu kuutio Chrome-selaimessa.

4.4 Tietokanta

MongoDB on avoimen lähdekoodin dokumenttipohjainen tietokanta. Tietokanta on toteutettu C++-ohjelmointikielellä ja siitä löytyy versiot Windows-, Linux-, Mac OS- sekä Solaris-käyttöjärjestelmille. (Introduction to MongoDB; MongoDB Manual, Installation)

MongoDB:n tietokanta koostuu kokoelmista (Collection), jotka pitävät sisällään dokumentteja. Yksittäisen dokumentin sisältämä tieto koostuu avain-arvo pareista. MongoDB:n dokumentit tallennetaan BSON-formaatissa, jonka rakenne muistuttaa JSON-formaattia. (Hows et al. 2013, Ch. 3; BSON Specification)

Alla olevassa kuvassa (Kuva 4.23) on esimerkki BSON-formaatin mukaisesta dokumentista, jolla kuvataan henkilön tietoja. Jokaisella dokumentilla on oltava yksikäsitteinen _id-tunniste. MongoDB luo tunnisteen dokumentille automaattisesti, mutta sen voi määritellä myös itse luontivaiheessa. (Hows et al. 2013, Ch. 3) Pakollisen tunnisteen lisäksi henkilölle on määritetty nimi ja yhteystiedot. Dokumentit voivat sisältää myös sisäkkäisiä dokumentteja. Esimerkissä sisäinen dokumentti on `Contact`-avaimen alla oleva lista, joka sisältää henkilön puhelinnumeron ja sähköpostiosoitteen.

```
{
    "_id" : 54e46d6e0bc64235dc6b813a,
    "name" : "Erkki Esimerkki",
    "contact" :
    [
        "phone" : "+3581234567",
        "email" : "user@example.com"
    ]
}
```

Kuva 4.23 Esimerkki BSON-formaatin mukaisesta dokumentista.

MongoDB eroaa perinteisestä relaatiotietokannasta siinä, että kokoelmaan tallennettavien dokumenttien skeemaa ei tarvitse määritellä etukäteen. Samassa kokoelmassa voi olla tietorakenteeltaan erilaisia dokumentteja. Skeeman määrittely jää sovellustason tehtäväksi. MongoDB onkin hyvä tilanteisiin, jolloin tallennettavan tiedon muotoa ei tiedetä etukäteen, tai se voi vaihdella suuresti.

MongoDB tarjoaa metodeja, joiden avulla tietokannassa olevia tietoja voi hakea, lisätä, muokata ja poistaa. Tietokannan metodeja voi kutsua esimerkiksi MongoDB:n asennuksen mukana tulevan Javascript-pohjaisen mongo shell -työkalun avulla. (MongoDB Manual)

Hakuja varten MongoDB tarjoaa jokaiselle kokoelmalle `find`-metodin. Metodille annettavat parametrit ja esittelyt on kuvattu alla olevassa taulukossa (Taulukko 4.3).

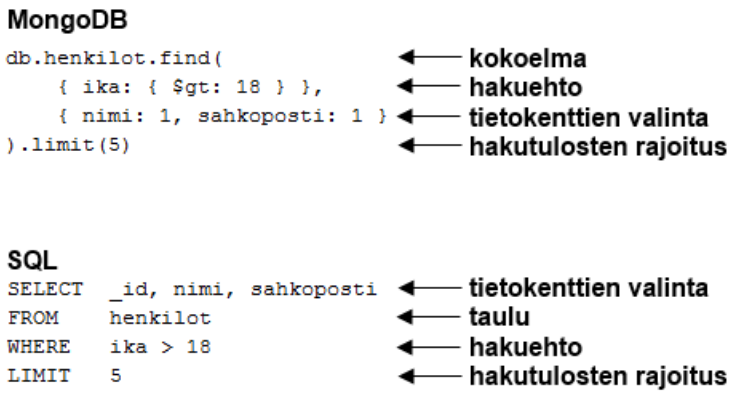
Taulukko 4.3 MongoDB find-metodin parametrit (MongoDB Manual).

```
db.collection.find(<criteria>, <projection>)
```

Parametri	
Criteria	Valinnainen. Sisältää haun parametrit.
Projection	Valinnainen. Määrittää tietokentät, joita halutaan palauttaa kyselyn tuloksena.
Paluuarvo	
Cursor	Kursori, jonka avulla iteroidaan läpi löydetyt dokumentit.

Metodin `Criteria`-parametreilla voidaan määritellä hakuehdot ja kriteerit, jotka löydetyn dokumentin on täytettävä. `Projection`-parametrien avulla voidaan rajoittaa kyselyn tuloksena palauttamat dokumentin tietokentät. Tällä tavoin voidaan pienentää asiakassovelluksen ja tietokannan välillä siirrettävän datan määrää. Metodin paluuarvona on kursori, jonka avulla löydetyt dokumentit voidaan iteroida läpi. Kursorille on mahdollista määrittää palautettavien dokumenttien määrä ja järjestystapa. (MongoDB Manual, Read Operations Overview)

MongoDB:n tarjoamien metodien syntaksi eroaa hieman relaatiokantojen SQL-kyselykielestä. Alla olevassa kuvassa (Kuva 4.24) on näkyvissä esimerkki MongoDB- ja SQL-kyselystä, joilla haetaan samoja tietoja. Molemmat kyselyt hakevat tietokannasta viisi henkilöä, joiden ikä-tietokentän arvo on yli 18. Haun palauttamat tietokentät on rajattu niin, että kyselyn tuloksena palautetaan vain henkilön nimi ja osoite.



Kuva 4.24 Esimerkki MongoDB- ja SQL-kyselystä.

MongoDB-haku kohdistuu henkilot-kokoelmaan ja SQL-kyselyssä henkilot nimiseen tauluun. MongoDB haussa suurempi kuin ehto voidaan määritellä vertailuoperaattorilla `$gt` (greater than). SQL-kyselyssä hakuehto määritellään `WHERE`-ehdon ”>”-vertailuoperaattorilla. Haun palauttamat tietokentät voidaan rajata MongoDB-kyselyssä `projection`-parametreissa määrittelemällä halutuille tietokentille arvo 1. SQL-kyselyssä tietokentät rajataan `SELECT`-lauseen parametreina. MongoDB kyselyn palauttamien dokumenttien määrää rajoitetaan `limit`-metodin parametrin arvolla, kun taas SQL-kyselyssä hakutulosten rajoittaminen tapahtuu `LIMIT`-määreellä.

Kuvan esimerkistä voidaan nähdä, että vaikka MongoDB:n haku eroaa syntaksiltaan ja termeiltään SQL-kyselystä, molemmat sisältävät monia samoja määrittelyjä. MongoDB:n hakutoiminnallisuudet sisältävät kuitenkin myös monia puutteita verrattuna SQL-kyselyihin. MongoDB ei esimerkiksi sisällä vierasavaimia tai muuta mekanismia, jolla voisi muodostaa automaattisesti liitoksia dokumenttien välille samaan tapaan kuin relaatiotietokannoissa. Vierasavainten päivittämisestä on huolehdittava itse sovellustasolla. Vierasavainten puuttumisen takia dokumentteja haettaessa voidaan joskus joutua tekemään useampi haku, jos haettavat dokumentit sijaitsevat eri kokoelmissa.

Dokumenttien lisääminen MongoDB-tietokannan kokoelmaan tapahtuu `insert`-metodilla. Alla olevassa taulukossa (Taulukko 4.4) on esitetty `insert`-metodin parametrit.

Taulukko 4.4 MongoDB insert-metodin parametrit (MongoDB Manual).

```
db.collection.insert(
   <document or array of documents>,
   { writeConcern: <document>, ordered: <boolean> }
)
```

Parametri	
Document	Kokoelmaan lisättävän dokumentin tai dokumenttien tiedot.
WriteConcern	Valinnainen. Asettaa kirjoitusoperaation tarkistuksen tason.
Ordered	Valinnainen. Jos true, useampien dokumenttien lisäys keskeytetään virheen sattuessa. Jos false, dokumenttien lisäämistä jatketaan, vaikka yksittäisen dokumentin lisäys epäonnistuisi.
Paluuarvo	
WriteResult, BulkWriteResult	Dokumentti, joka sisältää kirjoitusoperaation tuloksen. WriteResult palautetaan yksittäisen dokumentin lisäämiselle ja BulkWriteResult useita dokumentteja lisättäessä.

`Find`-metodin ensimmäisenä parametrina annetaan lisättävän dokumentin tiedot, joita voi olla myös useita. Parametrien `WriteConcern` ja `Ordered` avulla voidaan säätää kirjoitusoperaation tarkistuksen tasoa ja toimintaa virhetilanteen sattuessa. Metodi palauttaa dokumentin, joka sisältää kirjoitusoperaation tuloksen. Palautettavan dokumentin tyyppi vaihtelee sen mukaan onko lisätty yksi vai useampi dokumentti kerralla. (MongoDB Manual)

Alla olevassa kuvassa (Kuva 4.25) on esimerkki `insert`-metodin käytöstä. Esimerkissä lisätään henkilöä kuvaava dokumentti henkilot-kokoelmaan. Dokumentti sisältää tietokentät nimi, ikä ja sähköposti. MongoDB luo lisäysvaiheessa näiden tietokenttien lisäksi dokumentille automaattisesti `_id`-tunnisteen.

```
db.henkilot.insert(
    {
        nimi: "Erkki Esimerkki",
        ika: "32",
        sahkoposti: "user@example.com"
    }
)
```

Kuva 4.25 Esimerkki dokumentin lisäämisestä MongoDB-kokoelmaan.

Yksittäisten dokumenttien koko on rajoitettu 16 megatavuun. Tällä on pyritty rajoittamaan tietokannan muistin käyttöä. Suurempien dokumenttien ja tiedostojen tallennusta varten MongoDB tarjoaa GridFS tallennusmekanismin. GridFS käyttää hyväkseen tietokannasta oletuksena löytyviä files- ja chunks-kokoelmia. Files-kokoelmaan tallennetaan tiedoston metatiedot ja chunks-kokoelmaan tiedoston varsinainen binäärimuotoinen data.

Koska GridFS:n avulla tallennetut dokumentit eivät ole BSON-formaatissa, ei se myöskään sisällä samanlaisia hakuominaisuuksia kuin normaaleihin kokoelmiin tallennetut dokumentit. (MongoDB Manual, Data Models; Hows et al. 2013, Ch. 3)

Dokumenttien muokkaaminen tapahtuu `update`-metodin avulla. Alla olevassa taulukossa (Taulukko 4.5) on esitetty `update`-metodin parametrit.

Taulukko 4.5 MongoDB update-metodin parametrit.

```
db.collection.update(
   <query>,
   <update>,
   {
     upsert: <boolean>,
     multi: <boolean>,
     writeConcern: <document>
   }
)
```

Parametri	
query	Päivitettävän dokumentin hakuehdot.
update	Päivitettävät tiedot.
upsert	Valinnainen. Oletusarvo false. Jos true luodaan uusi dokumentti, mikäli hakuehtoja vastaavaa dokumenttia ei löydy.
multi	Valinnainen. Oletusarvo false. Jos true päivitetään kaikki hakuehtoja vastaavat dokumentit.
writeConcern	Valinnainen. Asettaa kirjoitusoperaation tarkistuksen tason.
Paluuarvo	
WriteResult	Dokumentti, joka sisältää päivitysoperaation tiedot ja tuloksen.

Päivitettävän dokumentin hakuehdot annetaan `query`-parametrin avulla. Päivitettävät tiedot asetetaan `update`-parametreissa. Dokumentti voidaan korvata kokonaan tai vaihtoehtoisesti päivittää vain halutut tietokentät. Dokumentin päivityslogiikkaa voidaan säätää käyttämällä `upsert`- ja `multi`-parametreja. `Upsert`-parametrin avulla päivityksen yhteydessä voidaan luoda uusi dokumentti, jos hakuehtoja vastaavaa dokumenttia ei löydy. `Multi`-parametrin avulla voidaan päivittää useampi dokumentti kerralla, jos ne täsmäävät hakuehtoihin. Kirjoitusoperaation tarkistuksen tasoa voi säätää parametrilla `writeConcern`, kuten `find`-metodillakin. Metodin paluuarvona on `WriteResult`-dokumentti, joka sisältää tiedot päivitysoperaatiosta. Tietojen avulla voidaan päätellä operaation onnistuminen ja esimerkiksi se kuinka monta dokumenttia päivitettiin. (MongoDB Manual)

Seuraavalla sivulla olevassa kuvassa (Kuva 4.26) on esimerkki `update`-metodin käytöstä. Esimerkissä päivitetään `_id`-tunnisteella 1 olevan dokumentin sähköpostiosoite. Esimerkissä käytetyn `$set`-määritteen avulla on määritetty päivitettävät tietokentät

```
db.henkilot.update(
    { _id: 1 },
    { $set: { sahkoposti: "erkki@esimerkki.com" } },
    { upsert: true }
)
```

Kuva 4.26 Esimerkki MongoDB-kokoelman dokumentin päivittämisestä.

Dokumenttien poistaminen tapahtuu `remove`-metodin avulla. Seuraavalla sivulla olevassa taulukossa (Taulukko 4.6) on esitetty metodin parametrit.

Taulukko 4.6 MongoDB remove-metodin parametrit.

```
db.collection.remove(
   <query>,
   {
     justOne: <boolean>,
     writeConcern: <document>
   }
)
```

Parametri	
query	Poistettavan dokumentin hakuehdot.
justOne	Valinnainen. Oletuksen false. Jos true, poistetaan vain yksi hakuehtoa vastaava dokumentti.
writeConcern	Valinnainen. Asettaa kirjoitusoperaation tarkistuksen tason.
Paluuarvo	
WriteResult	Dokumentti, joka sisältää poisto-operaation tuloksen.

Metodin `query`-parametrien avulla annetaan hakuehdot, joilla poistettava dokumentti löydetään. Dokumentin poistoon liittyviä ehtoja voidaan asettaa parametrilla `justOne`. Parametrin avulla voidaan määrittää poistetaanko kaikki löydetyt dokumentit vai pelkästään ensimmäiseksi löydetty. Kirjoitusoperaation tarkistuksen taso asetetaan parametrilla `writeConcern`. Metodin paluuarvona on `WriteResult`-dokumentti, joka sisältää tiedot poisto-operaatiosta, joiden avulla voidaan päätellä onnistuiko poistaminen ja kuinka monta dokumenttia tietokannasta poistettiin. (MongoDB Manual)

Alla olevassa kuvassa (Kuva 4.27) on esimerkki dokumentin poistamisesta. Esimerkissä poistetaan henkilot-kokoelmasta kaikki ne dokumentit, joissa ika-tietokentän arvo on alle 18.

```
db.henkilot.remove(
    { ika: { $lt: 18 } }
)
```

Kuva 4.27 Esimerkki dokumentin poistamisesta MongoDB-kokoelmasta.

MongoDB tarjoaa ohjelmointikirjastot useimmille ohjelmointikielille, kuten C, C++, C#, Java, Perl, PHP ja Python. Ohjelmointikirjastot sisältävät ajurin, jonka avulla eri ohjelmointikielillä toteutetut sovellukset voivat kommunikoida MongoDB-tietokannan kanssa. Ohjelmointikirjastot tarjoavat myös rajapinnan, joiden avulla asiakassovellukset pääsevät käyttämään tietokannan kokoelman haku-, lisäys-, muokkaus- ja poisto-metodeja. (MongoDB Manual)

Alla olevassa kuvassa (Kuva 4.28) on esimerkki C#-ohjelmointikielellä toteutetusta MongoDB:n C#-ohjelmointikirjastoa käyttävästä `MongoDBConnector`-luokasta. Luokan avulla voidaan ottaa yhteys MongoDB-tietokantaan ja muokata sen sisältämää persons-kokoelmaa.

```
public class MongoDBConnector
{
    private MongoCollection<PersonModel> _persons;

    // Luo yhteys tietokantaan.
    public MongoDBConnector()
    {
        MongoClient client = new MongoClient("mongodb://localhost"); // Asetetaan tietokannan yhteysosoite.
        MongoServer server = client.GetServer(); // Luodaan yhteys tietokantaan.
        MongoDatabase database = server.GetDatabase("example"); // Haetaan referenssi example-tietokantaan.
        _persons = database.GetCollection<PersonModel>("persons"); // Haetaan referenssi persons-kokoelmaan.
    }

    // Hae kaikki tietokannasta löytyvät henkilöt.
    public IEnumerable<PersonModel> GetAll()
    {
        return _persons.FindAll();
    }

    // Hae henkilö id:n perusteella.
    public PersonModel GetItem(string id)
    {
        IMongoQuery query = Query.EQ("_id", id); // Luodaan query-parametri _id-tunnisteelle.
        return _persons.Find(query).FirstOrDefault();
    }

    // Lisää henkilö tietokantaan
    public bool AddItem(PersonModel person)
    {
        person.Id = ObjectId.GenerateNewId().ToString(); // Luodaan uusi id-tunniste.
        WriteConcernResult result = _persons.Insert(person); // Lisätään tietokantaan insert-metodilla.
        return result.Ok;
    }

    // Päivitä henkilön tiedot
    public bool UpdateItem(PersonModel person)
    {
        var query = Query<PersonModel>.EQ(e => e.Id, person.Id); // Luodaan query-parametri id-tunnisteelle.
        // Päivitetään henkilön username- ja email-tietokentät.
        var update = Update<PersonModel>.Set(e => e.UserName, person.UserName).Set(e => e.Email, person.Email);
        WriteConcernResult result = _persons.Update(query, update); // Päivitetään tietokantaan update-metodilla.
        return result.UpdatedExisting;
    }

    // Poista henkilö tietokannasta.
    public bool RemoveItem(string id)
    {
        IMongoQuery query = Query.EQ("_id", id); // Luodaan query-parametri id-tunnisteelle.
        WriteConcernResult result = _persons.Remove(query); // Poistetaan tietokannasta remove-metodilla.
        return result.DocumentsAffected == 1;
    }
}
```

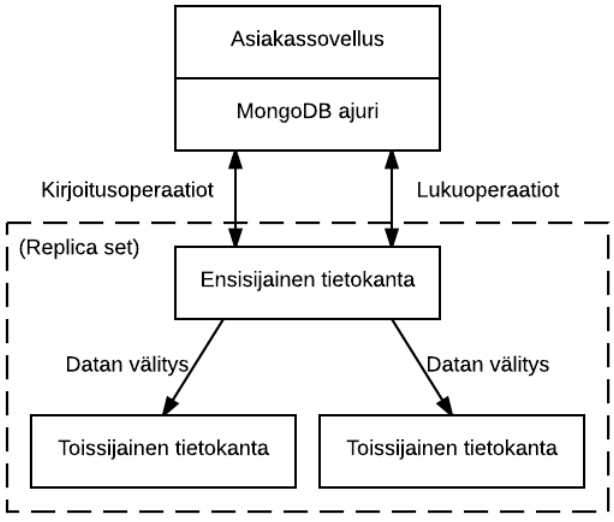
Kuva 4.28 Esimerkki C#-ohjelmointikielellä toteutetusta luokasta, jonka avulla voidaan ottaa yhteys MongoDB-tietokantaan.

`Persons`-kokoelma koostuu tavallisena C#-luokkana määritellyistä `PersonModel`-olioista. MongoDB:n tarjoama ohjelmointikirjasto osaa muuntaa C#-luokat automaattisesti BSON-muotoon ja takaisin olioiksi sarjallistamisen avulla.

Yhteys tietokantaan muodostetaan luokan rakentajassa ohjelmistokirjaston tarjoamien `MongoClient-`, `MongoServer-` ja `MongoDataBase-`luokkien avulla. Ohjelmointikirjaston käyttäjän ei tarvitse erikseen huolehtia yhteyden katkaisusta, vaan ohjelmointikirjasto pitää huolen siitä automaattisesti. Rakentajassa haetaan myös referenssi `persons`-kokoelmaan `MongoDBCollection`-luokan avulla, joka tallennetaan luokan yksityiseen muuttujaan.

Hakuja varten `MongoDBConnector`-luokka sisältää metodit, joiden avulla voidaan hakea persons-kokoelman kaikki dokumentit tai yksittäinen dokumentti id:n perusteella. `AddItem`-metodin avulla voidaan lisätä uusi henkilö `persons`-kokoelmaan. Dokumentin tietoja voidaan päivittää `UpdateItem`-metodin avulla. Dokumenttien poistaminen kokoelmasta tapahtuu luokan tarjoaman `RemoveItem`-metodin avulla.

Tietokannan varmuuskopiointia varten MongoDB tarjoaa replikointi-toiminnon (replication). Alla olevassa kuvassa (Kuva 4.29) on esitetty MongoDB:n replikoinnin idea. Replikointijoukko (Replica set) on joukko MongodB-tietokantojen instansseja, jotka pitävät sisällään saman datan.



Kuva 4.29 MongoDB:n replikoinnin idea (MongodB Manual, Replication Introduction).

Yksi joukon tietokannoista on ensisijainen ja muut toissijaisia. Ensisijainen tietokanta ottaa vastaan kaikki kirjoitusoperaatiot ja välittää ne edelleen toissijaisille tietokannoille, jotka kopioivat tiedot omaan tietokantaansa. Oletuksena ensisijainen tietokanta ottaa vastaan myös kaikki lukuoperaatiot, mutta ne voidaan tarvittaessa asettaa välittymään myös

toissijaisille tietokannoille. Tällä pyritään varmistamaan se, että lukuoperaatioiden palauttama data on aina varmasti ajan tasalla. Toissijaisten tietokantojen käyttö lukuoperaatioille voisi olla perusteltua esimerkiksi niissä tilanteissa, joissa halutaan parantaa lukuoperaatioiden tehokkuutta ja tietokannan sisältämän datan ajantasaisuus ei ole kriittistä. Tällöin lukuoperaatioihin voitaisiin käyttää aina asiakassovellusta lähinnä olevaa tietokantaa. (MongoDB Manual, Replication Introduction)

Ensisijaisen tietokannan vikaantuessa suoritetaan valintaprosessi, jonka aikana toissijaiset tietokannat äänestävät joukostaan uuden ensisijaisen tietokannan. Valittavaan ensisijaiseen tietokantaan on mahdollista vaikuttaa prioriteetti-asetuksella, jolla voidaan painottaa valintaa tai estää joidenkin tietokantojen valitseminen ensisijaiseksi. Valintaprosessin aikana tietokantaan ei voida kirjoittaa, joten se pyritään tekemään mahdollisimman nopeasti. Tätä varten joukon jäsenten välisiä yhteyksien vasteaikoja tarkkaillaan. Joukon jäsenet lähettävät toisilleen viestejä kahden sekunnin välein, joihin on vastattava 10 sekunnin kuluessa. Jos viesteihin ei kuulu vastausta, merkitään kyseinen joukon jäsen saavuttamattomaksi, eikä se enää voi osallistua valintaprosessiin. (MongoDB Manual, Replication Introduction)

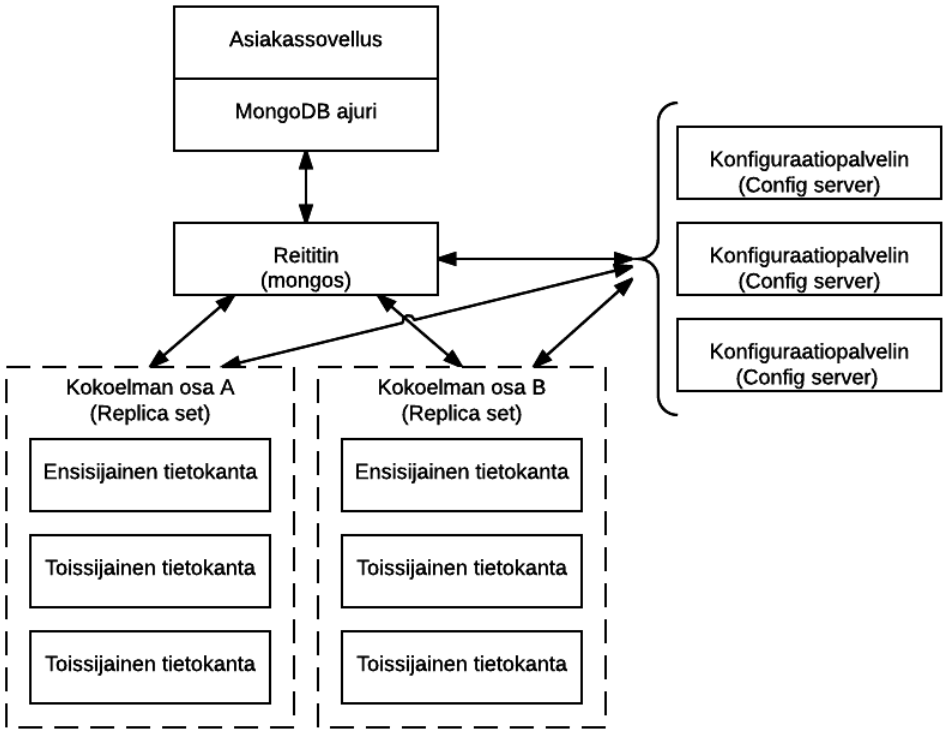
Replikoinnin avulla voidaan parantaa tietokannan saatavuutta, koska se tarjoaa keinon toipua tilanteista, joissa yksi tietokantapalvelimista vikaantuu. Toissijaisia tietokantoja voidaan käyttää hyväksi myös tiedon varmuuskopioinnissa ja raportoinnissa. (MongoDB Manual, Replication Introduction)

MongoDB tietokannan kokoelmia voidaan tarvittaessa hajauttaa MongoDB:n tarjoaman sharding toiminnon avulla. Tietokannan kokoelma hajautetaan valitsemalla kokoelman dokumenttien tietokentistä sopiva avain, joka on olemassa kaikilla dokumenteilla. Tätä avainta kutsutaan nimellä shard key. MongoDB jakaa olemassa olevat avaimet arvoalueisiin, joista yksi arvoalue vastaa aina yhtä tietokantaa. Arvoalueiden lukumäärä määräytyy siis käytössä olevien tietokantojen perusteella. Tietokentän arvoa voidaan käyttää sellaisenaan tai se voidaan ajaa hash-funktion läpi, jolloin voidaan varmistua dokumenttien jakautumisesta tasaisemmin eri tietokantoihin. Sharding toiminnon avulla hajautetun kokoelman arkkitehtuuri on näkyvillä seuraavalla sivulla olevassa kuvassa (Kuva 4.30). (MongoDB Manual, Sharding)

Arkkitehtuurin perusosat ovat reititin, konfiguraatiopalvelimet ja kokoelman osat, joihin kokoelma on hajautettu. Reititin on MongoDB:n mukana tuleva mongos-sovellus, jonka tehtävänä on vastaanottaa asiakassovelluksen tekemät luku- ja kirjoitusoperaatiot ja välittää ne edelleen kokoelman osille. Reititin lähettää myös tietokantojen vastaukset takaisin asiakassovellukselle. MongoDB suosittelee, että reitittimiä olisi tuotantokäytössä vähintään kaksi. Tällöin voidaan toipua reitittimen hajoamisesta sekä jakaa yksittäisen reitittimen käsittelemää datan määrää pienemmäksi. (MongoDB Manual, Sharding)

Reititin käyttää dokumenttien löytämisessä apuna MongoDB:n konfiguraatiopalvelimia, jotka sisältävät metatietoa kokoelman osista. Konfiguraatiopalvelimet pitävät sisällään tiedon esimerkiksi siitä miten dokumentit on jaoteltu sharding key:n avulla eri kokoelmien tietokantoihin. MongoDB:n suosituksen mukaan tuotantokäytännössä konfiguraatiopalvelimia tulisi olla vähintään kolme, koska tällöin voidaan toipua vikatilanteista. Reititin ei osaa ottaa yhteyttä kokoelmaan, jos yhteys konfiguraatiopalvelimelle katkeaa. (MongoDB Manual, Config servers)

Kokoelman data on hajautettu replikointijoukkojen (Replica set) avulla. Replikointijoukot toimivat samalla tavalla kuin MongoDB:n replikointi toiminnon yhteydessä. Tällä pyritään saavuttamaan samoja etuja kuin replikointi toiminnollakin eli toipumaan virhetilanteista mahdollisimman nopeasti. (MongoDB Manual, Sharding)



Kuva 4.30 MongoDB sharding-toiminnon avulla hajautetun tietokannan arkkitehtuuri.

Tietokannan hajauttamisella voidaan saavuttaa monia etuja. Hajautetun tietokannan suorituskykyä on mahdollista parantaa helposti, koska käytössä olevia palvelimia voidaan lisätä tarvittaessa. Hajauttamisen avulla on myös mahdollista pienentää tietokannan laitteistokustannuksia. Yksittäisen palvelimen ei tarvitse olla niin suorituskykyinen, koska käsiteltävän datan määrä on jaettu tasaisemmin useammalle palvelimelle. Tietokannan hajauttamisen avulla voidaan myös hoitaa nopeammin tilanteet, jossa tietokanta täytyy palauttaa varmuuskopioista. Varmuuskopiointi voidaan suorittaa rinnakkain kaikille kokoelman osille, jolloin toipumisaikaa voidaan huomattavasti lyhentää.

5. TIETOMALLIPALVELIMEN TOTEUTUS

Tässä luvussa esitellään kuinka valittuja toteutustekniikoita on käytetty tietomallipalvelimen toteutuksessa. Ensin käydään läpi järjestelmän arkkitehtuuri ja sen sisältämät komponentit yleisellä tasolla. Tämän jälkeen esitellään tietomallipalvelimen toiminnallisuudet. Seuraavaksi esitellään tietomallipalvelimen taustalla olevan tietokannan tietorakenne. Sitten käydään läpi tietomallipalvelimen käyttäjähallinnan toteutus. Lopuksi esitellään tietomallipalvelimen tarjoama rajapinta.

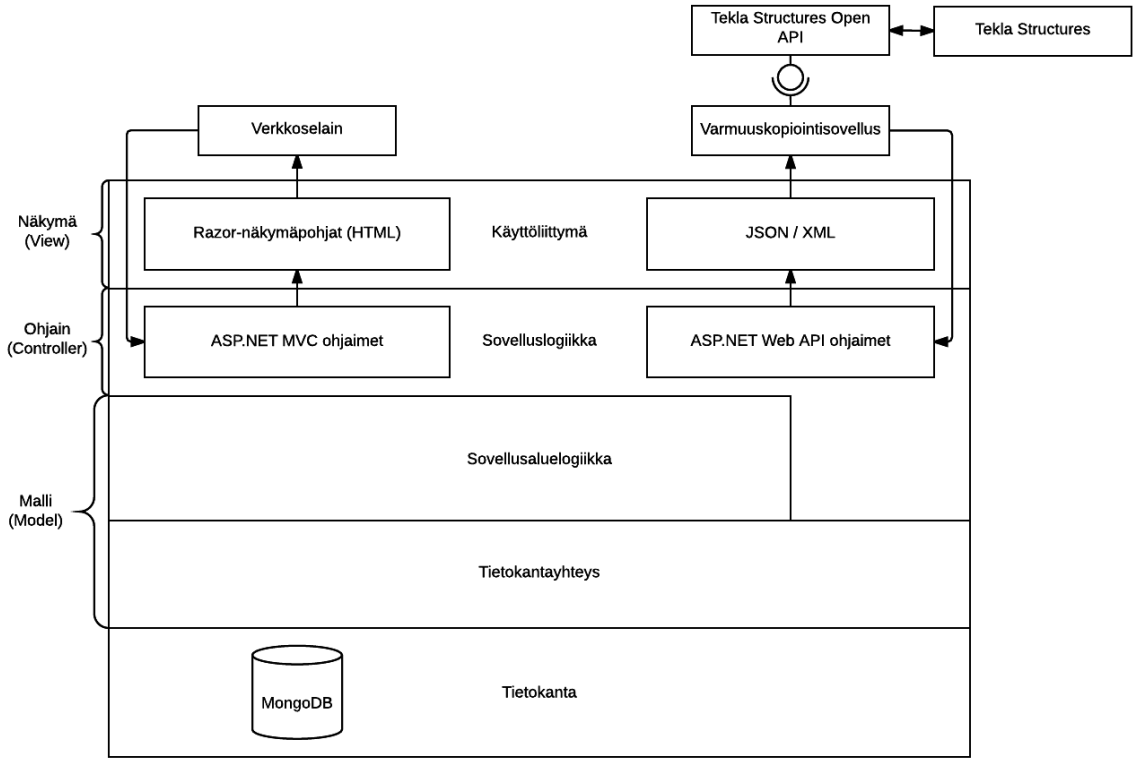
5.1 Järjestelmän arkkitehtuuri

Tietomallipalvelimen toteutuksessa hyödynnetään kahta eri sovelluskehystä. ASP.NET MVC-sovelluskehystä käytetään web-sovelluksen toteuttamiseen, joka tarjoaa selaimella käytettävän käyttöliittymän. ASP.NET Web API-sovelluskehystä käytetään REST-arkkitehtuurityylin mukaisen web-sovelluspalvelun luomiseen, joka tarjoaa rajapinnan varmuuskopioiden palautukseen käytetylle työpöytäsovellukselle. Käytettävät sovelluskehykset eivät ole suoraan yhteensopivia kaikilta osin, mutta molemmat ohjaavat kerrosarkkitehtuurin käyttämiseen. Kerrosarkkitehtuuria voidaankin hyödyntää myös sovelluskehysten yhdistämisessä. Seuraavan sivun kuvassa (Kuva 5.1) on esitetty tietomallipalvelimen arkkitehtuuri.

Arkkitehtuurin ylimmällä tasolla on näkymäkerros. Web-sovelluksen käyttöliittymä muodostetaan HTML:n avulla. Näkymän muodostuksessa käytetään Razor-merkkauskielen avulla luotuja näkymäpohjia, jotka näkymämoottori kääntää HTML-muotoon. REST-rajapinnan toteuttava web-sovelluspalvelu lähettää datan varmuuskopiointisovellukselle XML- tai JSON-muodossa. Sekä web-sovellus, että web-sovelluspalvelu käyttää asiakkaan ja palvelimen väliseen kommunikointiin HTTP-protokollaa.

Toiseksi ylimmällä tasolla on sovelluslogiikan sisältävä kerros, joka pitää sisällään ASP.NET MVC- ja ASP.NET Web API-sovelluksissa käytettävät ohjain-luokat. Ohjainluokkia käytetään välittämään asiakkaan lähettämät pyynnöt alimmille kerroksille ja edelleen alimpien kerrosten lähettämät vastaukset takaisin asiakkaalle.

Molemmat sovelluskehykset hyödyntävät samoja malliluokkia ja tietokantakerrosta. Tämä johtuu siitä, että molempien sovelluskehysten tietosisältönä voidaan käyttää tavallisia .NET-luokkia. Samojen luokkien hyödyntäminen helpottaa sovelluskehitystä, koska saman toiminnallisuuden sisältäviä komponentteja ei tarvitse toteuttaa kahteen kertaan molemmille sovelluskehyksille. Samalla helpotetaan myös järjestelmän ylläpitoa ja korjausten tekemistä, koska muutokset tarvitsee tehdä vain yhteen paikkaan.



Kuva 5.1 Tietomallipalvelimen arkkitehtuuri

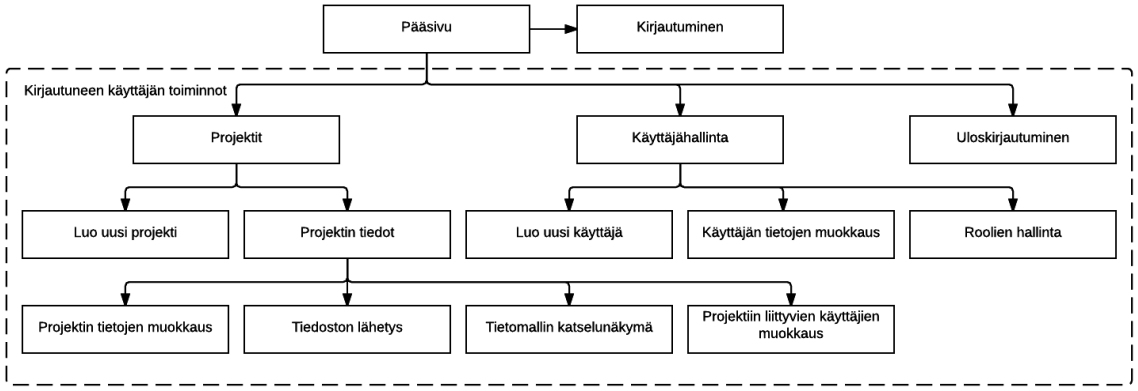
Sovelluslogiikkakerroksen alapuolella on sovellusaluelogiikkakerros. Sovellusaluelogiikkakerros pitää sisällään sovelluksen tietorakennetta kuvaavat luokat. Samassa kerroksessa sijaitsevat myös sovelluskehystä laajentavat kolmannen osapuolen kirjastot IFC Engine ja Hangfire. IFCEngine-kirjaston avulla suoritetaan tietomallipalvelimelle lähetettyjen IFC-tiedostojen parsinta ja muuntaminen sovelluksen sisäisen tietorakenteen mukaiseksi. Hangfire-kirjastoa käytetään hyväksi tietomallipalvelimella suoritettavien pitkäaikaisten tehtävien hallinnassa.

Toiseksi alimmalla tasolla on tietokannan abstrahoiva tietokantayhteyskerros. Tietokantayhteyskerros tarjoaa palvelut, joiden avulla ylemmät kerrokset voivat hakea ja muokata tietokannan sisältämiä tietoja. Alimmalla tasolla on tietokantakerros, joka sisältää järjestelmän tietosisällön, kuten tietomallit ja projekteihin liittyvät tiedot sekä käyttäjätiedot. Tietokannan rakenne on kuvattu tarkemmin kohdassa 5.3.

5.2 Toiminnallisuudet

Tietomallipalvelimen tarjoamia toimintoja käytetään selaimen kautta. Seuraavalla sivulla olevassa kuvassa (Kuva 5.2) on kuvattu tietomallipalvelimen selaimella käytettävän käyttöliittymän navigointikaavio ja mahdolliset siirtymät eri sivuille.

Ylimmällä tasolla on pääsivu, josta käyttäjä pääsee navigoimaan alemman tason toimintoihin. Ylimmältä tasolta käyttäjä pääsee kirjautumissivulle, tarkastelemaan omia projekteja tai käyttäjähallintaan. Osa toiminnoista vaatii käyttäjältä kirjautumisen. Kyseiset toiminnot on rajattu kuvassa katkoviivalla.

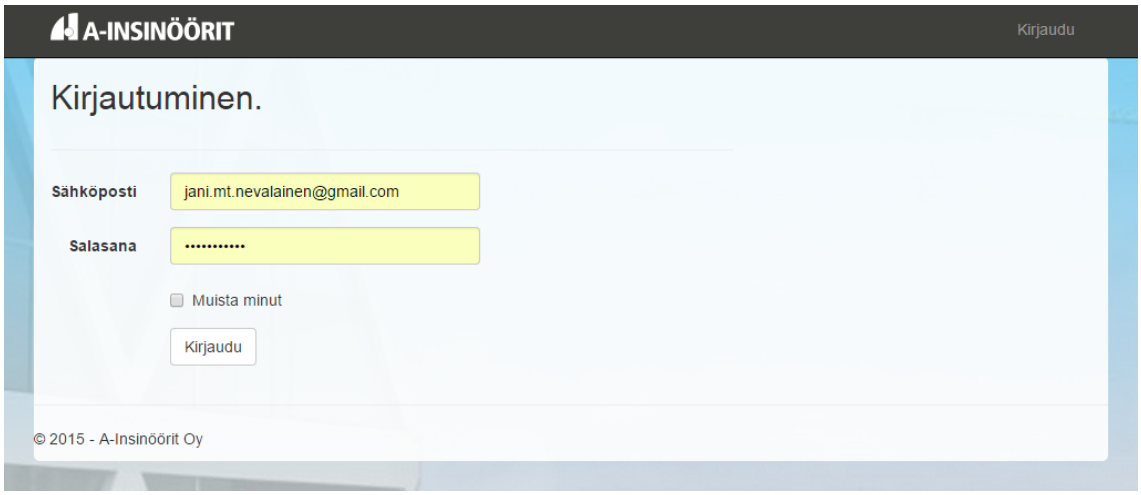


Kuva 5.2 Tietomallipalvelimen navigointikaavio

Osa toiminnoista, kuten Käyttäjähallinta ja sen alapuolella olevat toiminnot eivät ole näkyvissä kaikille kirjautuneille käyttäjille. Käyttäjähallintaan pääsee vain sivuston Admin-roolissa olevat käyttäjät. Käyttäjähallinnan roolit on kuvattu tarkemmin kohdassa 5.4.

5.2.1 Kirjautuminen

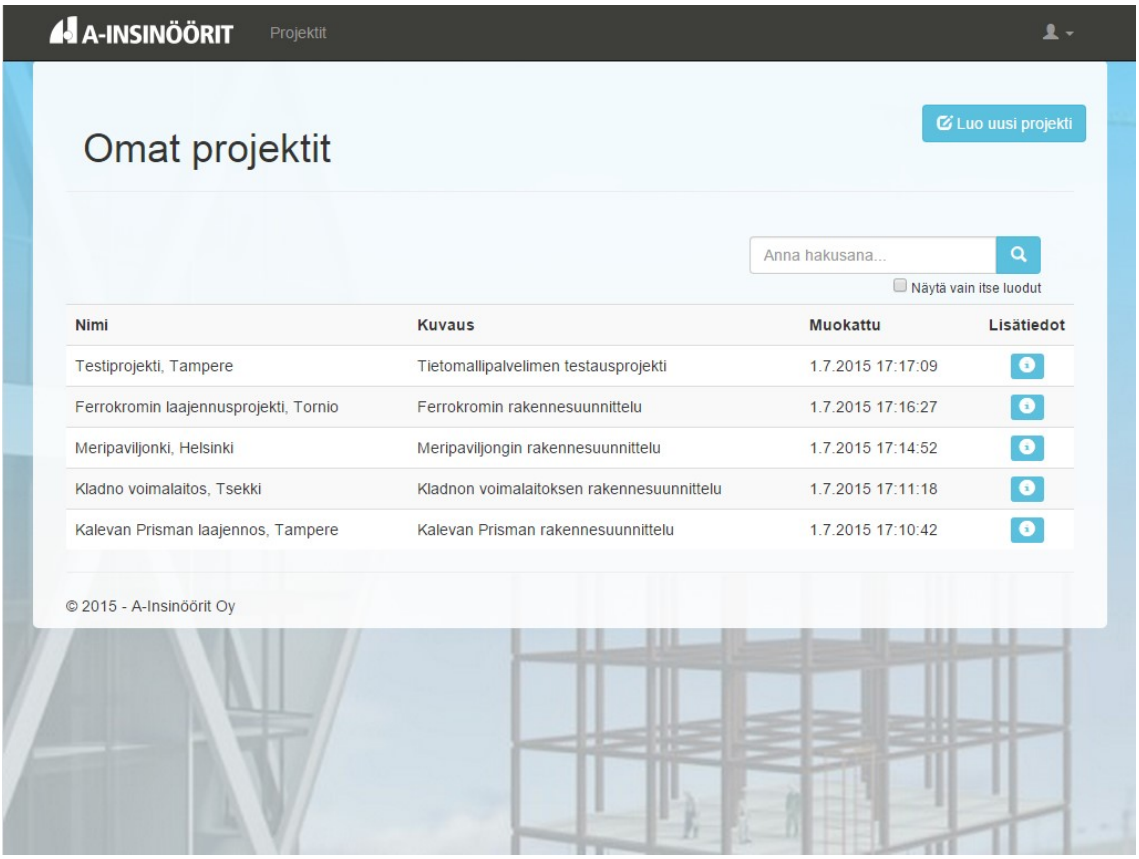
Kirjautumaton käyttäjä pääsee palvelun etusivulta vain kirjautumissivulle. Kirjautumissivulla näytetään lomake, johon käyttäjä voi syöttää käyttäjätunnuksena käytettävän sähköpostiosoitteen, sekä salasanan. Kirjautumissivu on näkyvissä alla olevassa kuvassa (Kuva 5.3). Onnistuneen kirjautumisen jälkeen sivun yläreunan Kirjaudu-toiminto muuttuu Kirjaudu ulos-toiminnoksi, jonka avulla käyttäjä voi kirjautua palvelusta ulos.



Kuva 5.3 Tietomallipalvelimen kirjautumissivu.

5.2.2 Projektit

Projektit-sivulla näytetään listaus käyttäjän omista projekteista. Projekteista näytetään listassa vain osa projektin otsikkotiedoista, kuten nimi, kuvaus ja muokkauspäivämäärä. Yksittäisen projektin tarkempia tietoja pääsee tarkastelemaan klikkaamalla lisätiedot ikonia, jonka kautta pääsee Projektin tiedot-sivulle. Projektit-sivu on näkyvillä alla olevassa kuvassa (Kuva 5.4).



Kuva 5.4 Tietomallipalvelimen projektien listausnäkymä.

Näytettäviä projekteja voi halutessaan suodattaa näkymässä olevan hakukentän avulla. Haku hakee annetulla hakusanalla kaikista projektin otsikkokentistä. Käyttäjä voi halutessaan suodattaa näkyviin vain itse luomansa projektit asettamalla "Näytä vain itse luodut"-valintaruudun aktiiviseksi.

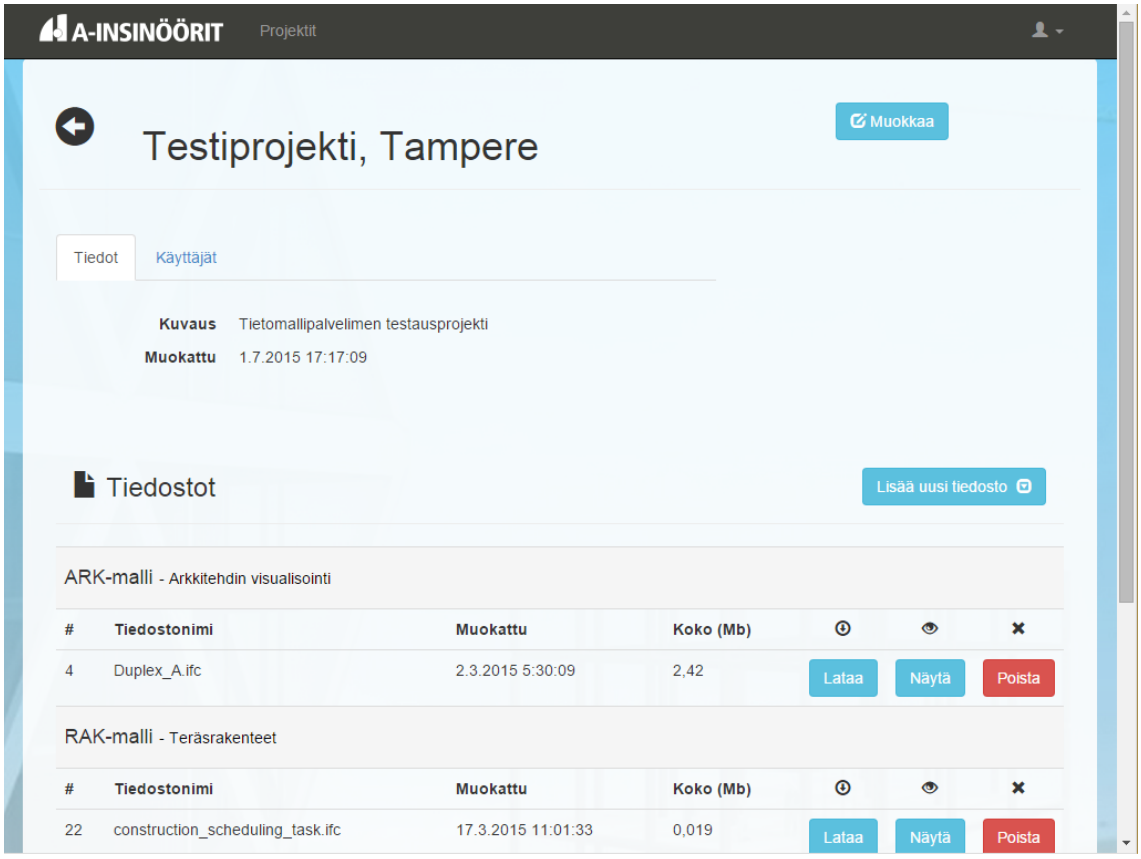
Projektit sivulta pääsee luomaan uusia projekteja Luo uusi projekti-toiminnolla. Toiminto ohjaa Projektin lisäys-sivulle. Projektin lisäys-sivulla näytetään projektien lisäyslomake. Lomakkeessa määritetään projektin otsikkotiedot, kuten nimi ja kuvaus. Lomakkeessa määritetään myös ne käyttäjät, jotka pääsevät tarkastelemaan projektin tietoja.

5.2.3 Projektin tiedot

Projektin tiedot-sivulla näytetään projektiin liittyvät yleistiedot, projektiin liitetyt käyttäjät sekä listauksen projektille lisätyistä IFC-tiedostoista ja niiden versiohistorian. Projektin tiedot-sivu on näkyvissä alla olevassa kuvassa (Kuva 5.5).

Projektin tiedot-sivulta pääsee muokkaamaan projektin tietoja, lisäämään uusia tiedostoja projektille sekä muokkaamaan projektiin liittyviä käyttäjiä. Projektin tiedot-sivulta käyttäjä pääsee myös tietomallin katselunäkymään.

Projektille lisättyjen IFC-tiedostojen listauksessa näytetään uusin tiedostoversio sekä sen yleistiedot, kuten tiedostonimi, kuvaus, lisäyspäivämäärä ja tiedoston koko. Samalla rivillä näytetään myös toiminnot Lataa, Näytä ja Poista. Lataa-toiminnon avulla voi ladata alkuperäisen IFC-tiedoston. Näytä-toiminnon avulla pääsee tietomallin katselunäkymään. Poista-toiminnon avulla voi poistaa kyseisen IFC-tiedoston version palvelimelta.

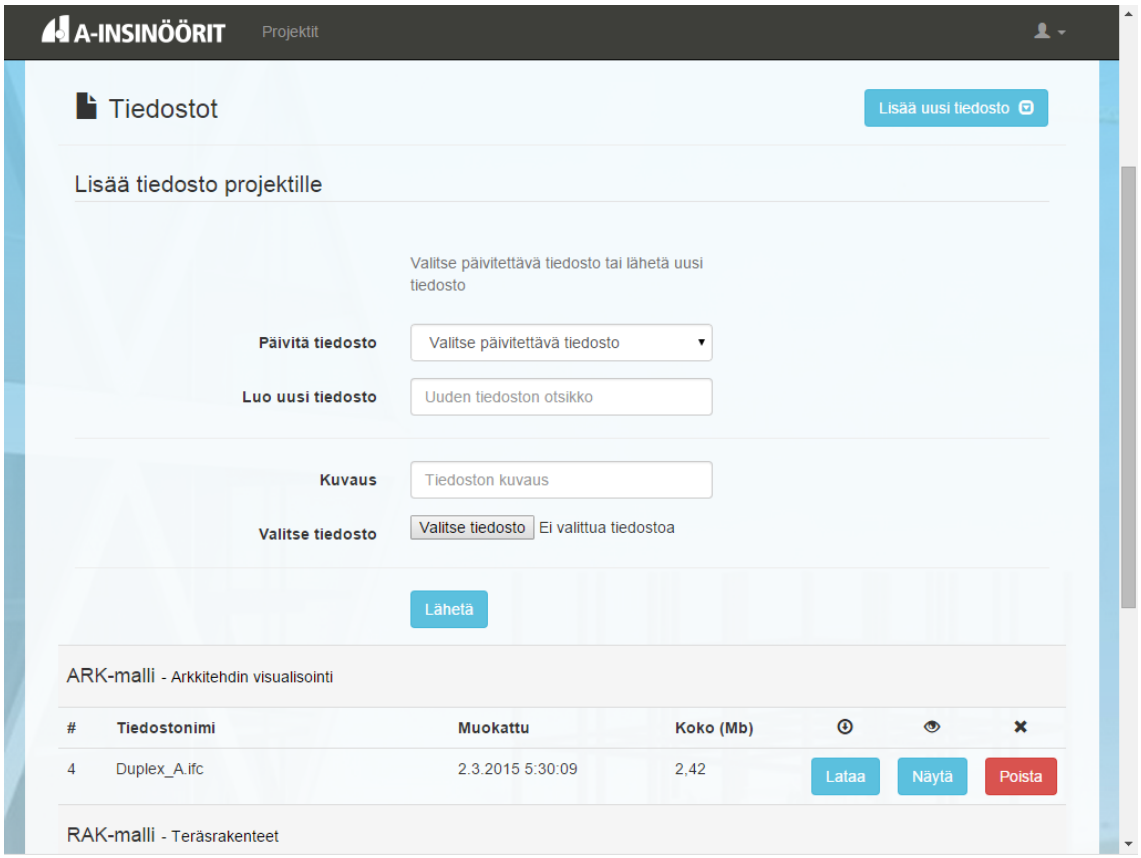


Kuva 5.5 Projektin tiedot näkymä.

IFC-tiedoston versiohistorian saa näkyviin klikkaamalla listauksen riviä, joka laajentaa näkyviin listauksen aiemmista versiosta. Myös aiemmin tallennetuilla versioilla on käytettävissä samat toiminnot kuin uusimmalla tiedostoversiolla.

Uusia IFC-tiedostoja voi lähettää sivulta löytyvän Lisää uusi tiedosto-toiminnon avulla. Toiminto avaa esiin lomakkeen, jolla voi lähettää uuden tiedoston, tai lisätä uuden version

aiemmin lähetettyyn IFC-tiedostoon. Tiedostojen lähetykseen käytettävä lomake on näkyvissä alla olevassa kuvassa (Kuva 5.6).



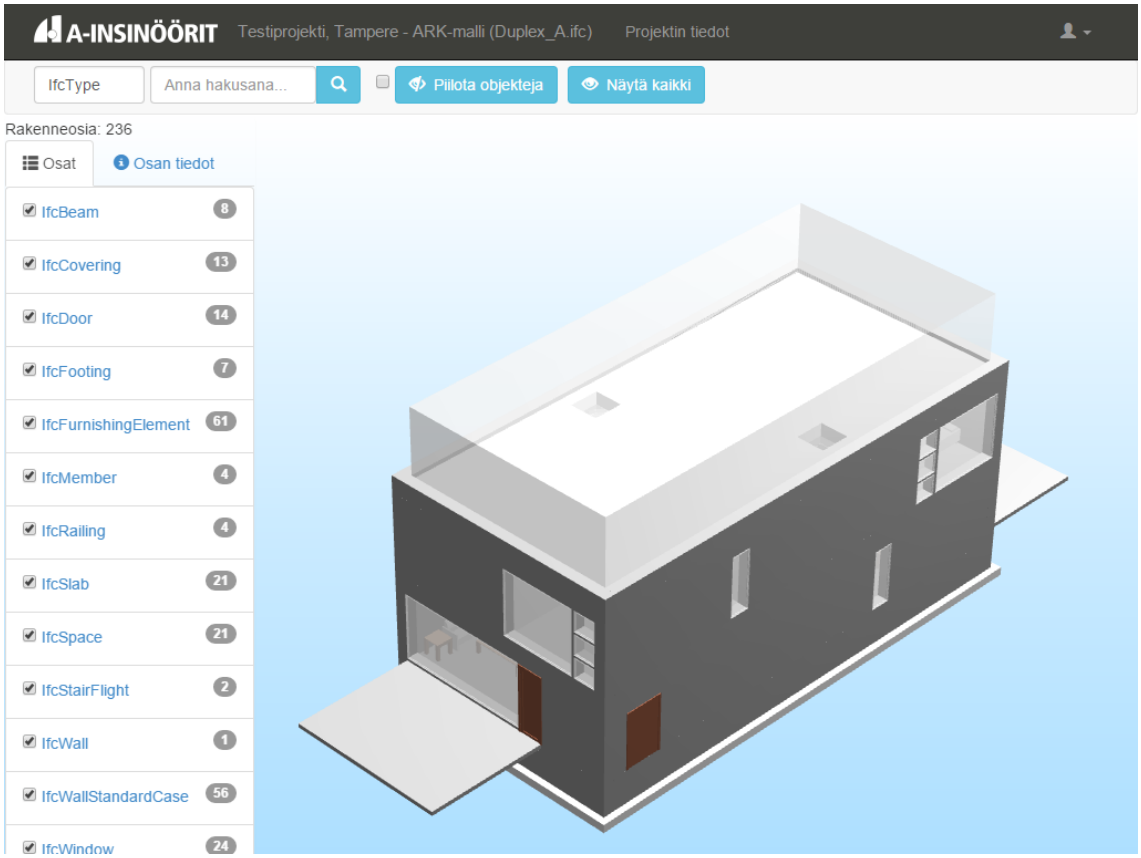
Kuva 5.6 Tiedostojen lisääminen projektille.

Projektin tietoja voi muokata näkymästä löytyvällä Muokkaa-toiminnolla. Muokkaa-toiminto avaa projektin muokkaus-sivun, jonka avulla voi muuttaa projektin otsikkotietoja, tai sille lähetettyjen tietomalli-tiedostojen otsikkotietoja. Muokkaus-sivun kautta voi tämän lisäksi poistaa tai lisätä projektiin lisättyjä käyttäjiä. Myös projektin poistaminen tapahtuu muokkaus-näkymän kautta. Poista-toiminnon avulla voidaan poistaa projekti ja sen sisältämät tietomallitiedostot palvelimelta.

5.2.4 Tietomallin katselunäkymä

Tietomallin katselunäkymässä näytetään IFC-tiedoston sisältämien rakenneosien geometria 3D-mallina. Tietomallin tarkastelunäkymä on näkyvissä seuraavalla sivulla olevassa kuvassa (Kuva 5.7).

Näkymän yläreunassa on näkyvissä navigaatiopalkki, joka sisältää alkuperäisen IFC-tiedoston nimen, sekä linkin jolla pääsee takaisin projektin tietoihin. Näkymän vasemmassa reunassa olevalla Osat-välilehdellä näytetään tietomallin sisältämät osatyypit, sekä niiden lukumäärä. Osatyyppejä voi suodattaa näkyviin tai pois näkyvistä osatyypin vasemmalla puolella olevan valintaruudun avulla.



Kuva 5.7 Tietomallipalvelimen tietomallin katselunäkymä.

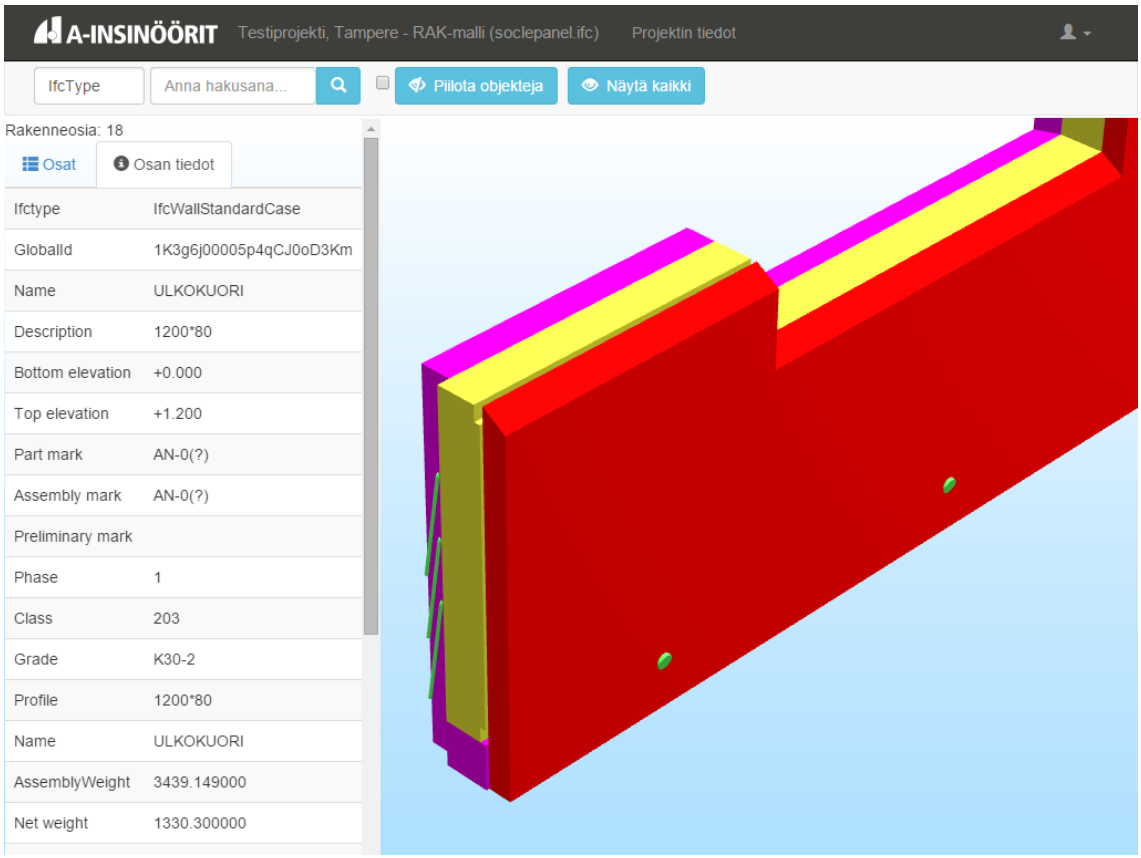
Yksittäisiä rakenneosia voi piilottaa näkymässä olevan Piilota objekteja-valintaruudun avulla. Valintaruudun ollessa aktiivinen 3D-mallista valitut rakenneosat piilotetaan näkyvistä. Piilotetut rakenneosat saa takaisin näkyviin 3D-malliin näkymässä olevan Näytä kaikki-toiminnon avulla.

Haluttuja rakenneosia voi suodattaa näkyviin myös Haku-toiminnolla, jota kuvataan suurennuslasi-ikonilla. Haun parametreina annetaan rakenneosan ominaisuustiedon nimi, sekä sen arvo. Esimerkiksi haulla, jonka ominaisuustiedon nimi on "Name" ja arvo "Ulkokuori", voidaan suodattaa näkyviin kaikki rakenneosat, joiden nimi vastaa kyseistä arvoa.

Haun avulla voidaan tarkistaa myös suunnittelun valmiusaste. Tekla Structures-suunnittelusovelluksessa suunnittelun valmiusaste voidaan määrittää asettamalla rakenneosille ominaisuustieto nimeltä "DESIGN_STATUS". Valmiusasteelle asetettavia arvoja on rajattu määrä ja ne ovat ennalta määritettyjä. Eri valmiusasteet saadaan siis selville käyttämällä haun ominaisuustietona kyseistä nimeä, sekä arvona haluttua suunnittelun valmiusastetta.

Yksittäisen rakenneosan ominaisuustiedot saa näkyviin klikkaamalla kyseistä osaa 3D-mallista. Valittu rakenneosa korostetaan punaisella värillä 3D-mallissa. Rakenneosan

ominaisuustiedot näytetään näkymän vasemmassa reunassa olevassa Osan tiedot-välilehdellä. Ominaisuustiedot listataan allekkain avain-arvo-pareina, jossa avaimena käytetään ominaisuuden nimeä ja arvona kyseisen ominaisuuden arvoa. Alla olevassa kuvassa (Kuva 5.8) on näkyvissä esimerkki valitun rakenneosan sisältämistä ominaisuustiedoista.



Kuva 5.8 Tietomallin katselunäkymä ja yksittäisen osan tiedot.

Tämän lisäksi välilehdellä näytetään rakenneosalle linkitetyt dokumentit. Linkin lisääminen tapahtuu Osan-tiedot välilehdeltä löytyvällä Lisää linkki-toiminnolla, joka avaa linkin lisäämisen tarkoitetun lomakkeen. Lomakkeessa annetaan linkin URI-osoite ja kuvaus, joka kertoo linkin sisällöstä.

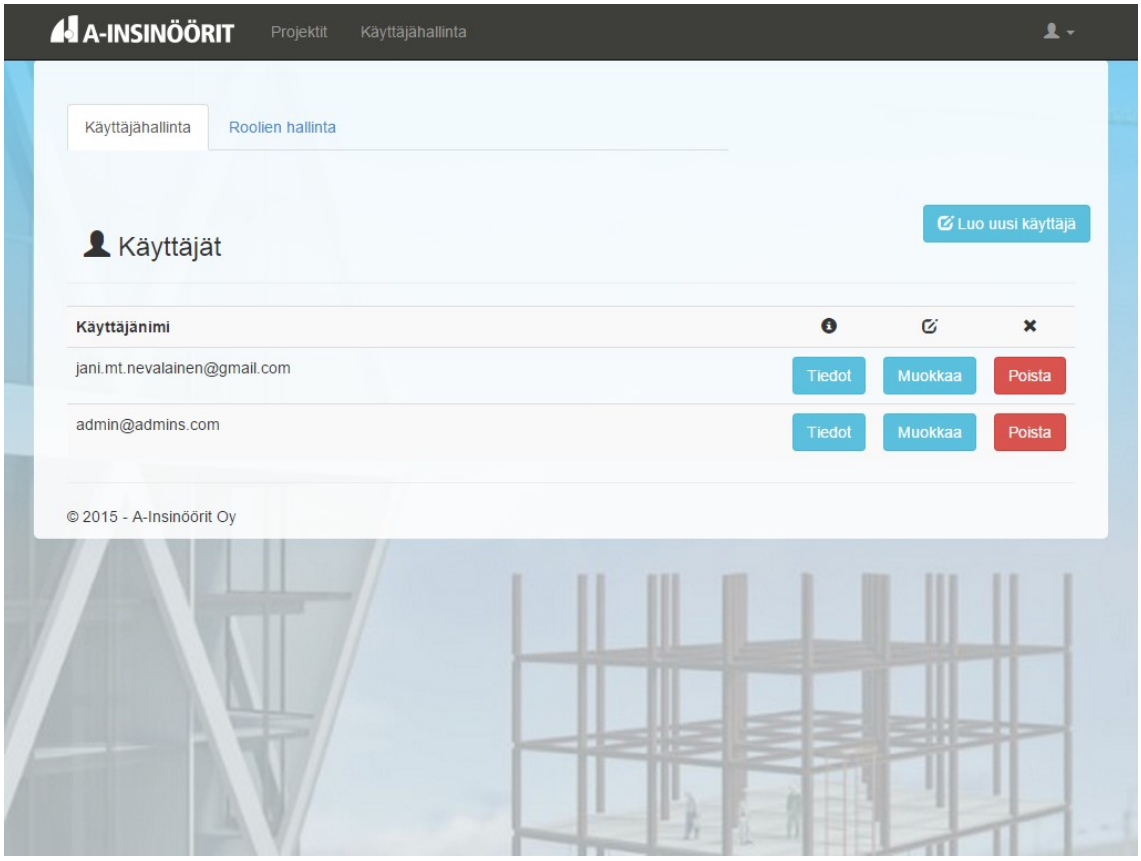
5.2.5 Käyttäjähallinta

Käyttäjähallinta-sivulla näytetään listaus tietomallipalvelimella olevista käyttäjistä. Käyttäjähallinta-sivulta pääsee myös käyttäjien lisäys sivulle sekä muokkaamaan yksittäisen käyttäjän tietoja. Tällä hetkellä käyttäjähallinta-sivulle pääsevät vain Admin-roolissa olevat käyttäjät. Käyttäjähallinta-sivu on näkyvissä seuraavalla sivulla olevassa kuvassa (Kuva 5.9).

Käyttäjien lisääminen järjestelmään tapahtuu näkymästä löytyvällä Luo uusi käyttäjä-toiminnolla. Toiminto avaa lomakkeen, johon syötetään käyttäjän käyttäjätunnuksena käytettävä sähköpostiosoite sekä muut yleistiedot. Lisäyksen yhteydessä määritetään myös käyttäjän rooli, joka voi olla sivuston ylläpitäjä, yrityksen sisäinen tai asiakas.

Sivuston ylläpitäjät pääsevät muokkaamaan tietomallipalvelimelta projekteja ja niihin liittyviä tiedostoja. Yrityksen sisäinen-roolissa olevat käyttäjät pääsevät muokkaamaan vain niitä projekteja, jotka he ovat itse luoneet, tai niitä projekteja joihin heidät on liitetty. Asiakas-roolissa olevat käyttäjät pääsevät katselemaan projekteja, joihin heidät on liitetty, mutta he eivät pääse muokkaamaan projektin tiedostoja.

Käyttäjän lisäämisen jälkeen uudelle käyttäjälle lähetetään sähköposti, jonka kautta voi määrittää palvelussa käytettävän salasanan. Vasta kun käyttäjä on määrittänyt salasanan, on käyttäjätunnus käytettävissä palvelussa.



Kuva 5.9 Tietomallipalvelimen käyttäjähallinta-näkymä.

Yksittäisen käyttäjän tarkempia tietoja voi tarkastella käyttäjälistauksessa löytyvällä Tiedot-toiminnolla. Toiminto avaa käyttäjän tiedot-sivun, jossa näytetään käyttäjään liittyvät yleistiedot, sekä tiedot käyttäjän rooleista.

Käyttäjien tietoja voi muokata käyttäjälistauksessa löytyvällä Muokkaa-toiminnolla. Muokkaa-toiminto avaa lomakkeen, jonka avulla käyttäjän tietoja tai roolia voi muuttaa.

Käyttäjien poistaminen tapahtuu käyttäjähallinta-sivun listauksesta löytyvällä Poista-toiminnolla.

Käyttäjähallinta-näkymässä olevan Roolien hallinta-välilehden kautta pääsee näkemään palvelimella olevat roolit. Tällä hetkellä tietomallipalvelimella olevat roolit ovat kiinteästi etukäteen määriteltyjä. Näkymän kautta uusia rooleja on mahdollista lisätä, mutta niiden huomioiminen tietomallipalvelimen toiminnallisuudessa vaatii muutoksia sovelluskoodiin.

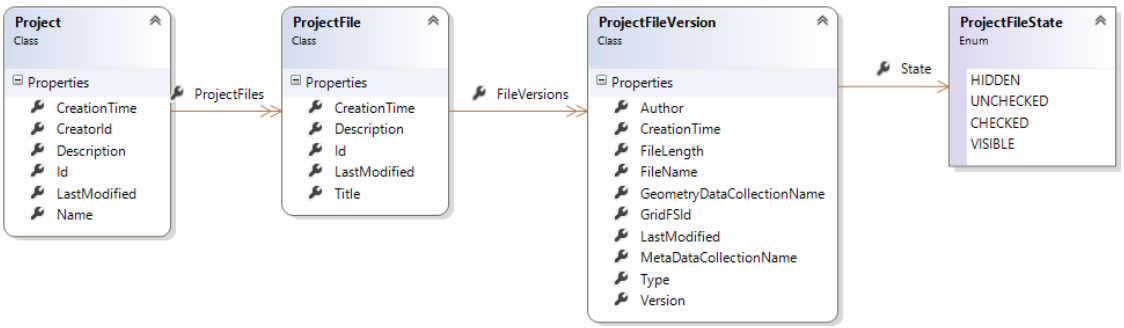
5.3 Tietokanta

Tietomallipalvelimen sisältämät projektien tiedot sekä niihin liittyvät tietomallitiedostot tallennetaan MongoDB-tietokantaan. Koska MongoDB:n tarjoama ohjelmointikirjasto osaa muuntaa C#-luokat automaattisesti BSON-muotoon ja takaisin olioiksi sarjallistamisen avulla, on useimpien kokoelmaan tallennettavien dokumenttien tietorakenne määritetty tavallisten C#-luokkien avulla.

Johtuen MongoDB-tietokannan dokumenttien 16MB:n kokorajoitteesta, tietomallitiedostoja ei voida tallentaa suoraan kokoelmaan yksittäisenä dokumenttina. Tämän vuoksi projektin ja siihen liittyvien tietomallitiedostojen tietoja on jouduttu hajauttamaan eri kokoelmiin. Kokoelmat on pyritty optimoimaan järkeviksi useimmin suoritettujen hakuoperaatioiden kannalta. Optimoinnilla on pyritty vähentämään haun sisältämän ylimääräisen datan määrää, sekä välttämään tilanteita, joissa dokumentteja tarvitsee käsitellä jälkikäteen ennen lähetystä asiakkaalle.

5.3.1 Projektien tietorakenne

Projektien tallentamiseen käytetään Projects-kokoelmaa. Alla olevassa kuvassa (Kuva 5.10) on kuvattu yksittäisen kokoelmaan tallennettavan luokan tietorakenne UML-kaavion avulla.



Kuva 5.10 Projektien tietorakenne.

Projektin tietorakenteen kuvaamisessa hyödynnetään `Project`-, `ProjectFile`- ja `ProjectFileVersion`-luokkia. `Project`-luokka pitää sisällään projektin yleisiä tietoja kuvaavat tietokentät. `Project`-luokalla on lista `ProjectFile`-olioista, joilla kuvataan projektille lisättyjen tietomallitiedostojen otsikkotietoja. `ProjectFile`-luokka pitää sisällään listan `ProjectFileVersion`-olioita, joita käytetään kuvaamaan tietomallitiedoston versioita. Jotta dokumenttien koko ei kasvaisi liian suureksi, järjestelmään tallennetaan vain 20 viimeisintä versiota.

`ProjectFileVersion`-luokka sisältää tiedoston versionumeron lisäksi, IFC-tiedostoon liittyviä metatietoja, kuten tiedoston koon, alkuperäisen nimen, muokkauspäivämäärän ja tiedoston tilan. Tiedoston tilaa kuvataan `ProjectFileState`-enum tietotyypin avulla. Tiedoston tilaa käytetään hyväksi tiedoston lähetysvaiheessa ilmaisemaan, onko kyseinen IFC-tiedosto tarkistettu ja sen sisältämät tiedot tallennettu tietokantaan kokonaisuudessaan. Tämän lisäksi luokka sisältää tiedot, joiden avulla saadaan selville alkuperäisen tietomallitiedoston sijainti sekä tietomallin meta- ja geometriadataa kuvaavien kokoelmien nimet, joissa niitä kuvaavat dokumentit sijaitsevat.

Tällä hetkellä meta- ja geometriadataa kuvaavien kokoelmien nimet ovat kiinteitä, mutta niiden avulla dokumentit voitaisiin tarvittaessa jakaa eri kokoelmiin esimerkiksi jokaiselle yksittäiselle projektille. Tällä voidaan parantaa suorituskykyä niissä tapauksissa, kun projektien lukumäärä tietokannassa lisääntyy.

Alkuperäisen IFC-tiedoston tallentamiseen käytetään MongoDB:n GridFS-kokoelmaa, joka on tarkoitettu tiedostojen tallentamiseen. Jokaisella GridFS-kokoelmaan tallennetulla tiedostolla on yksikäsitteinen `id`-tunniste, joka asetetaan `ProjectFileVersion`-luokan `GridFSId`-attribuutin arvoksi. GridFS-kokoelmaan tallennetuille tiedostoille ei ole kokorajoitetta. Rajoitteena on sen sijaan se, että GridFS-kokoelmaan tallennettujen tiedostojen sisälle ei voi suorittaa samanlaisia hakuja kuin normaaleihin kokoelmiin tallennetuille dokumenteille. Tässä tapauksessa kyseisestä rajoitteesta ei kuitenkaan ole haittaa, sillä IFC-tiedoston sisältämä meta- ja geometriadata tallennetaan erikseen eri kokoelmiin hakuja varten.

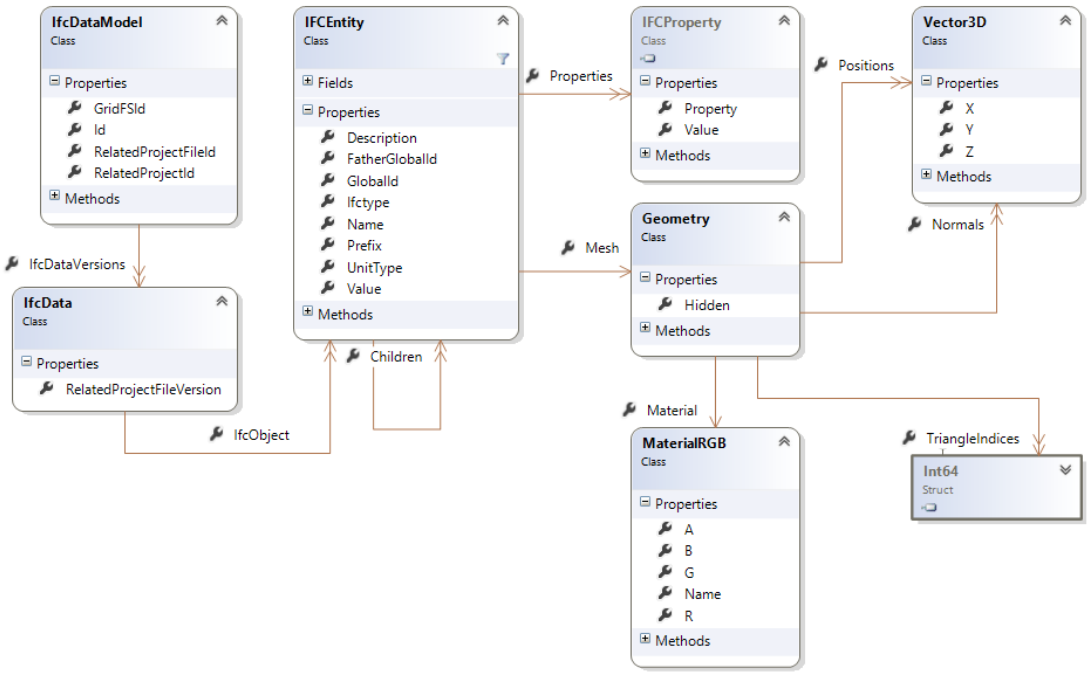
5.3.2 IFC-tietomallin meta- ja geometriatietojen tietorakenne

Tietomallipalvelimelle lähetettävät IFC-tiedostot parsitaan IFC Engine-kirjaston avulla ennen tallentamista tietokantaan. Tällä voidaan varmistua tiedoston oikeellisuudesta sekä hylätä virheelliset tiedostot ennen tallennusta. Parsinnan aikana IFC-tiedostosta erotetaan kunkin rakenneosan metatiedot ja geometria sekä alustetaan IFC-tiedoston sisältämää metadataa ja geometriaa kuvaavat luokat. IFC-mallin metatietojen ja geometrian kuvaamiseen käytetty tietorakenne on esitetty UML-kaavion seuraavalla sivulla olevassa kuvassa (Kuva 5.11).

Meta- ja geometriadatan säilömisessä käytetään apuna `IfcDataModel`-luokkaa. Jokaiselle IFC-tiedoston sisältämälle rakenneosalle luodaan meta- tai geometria-kokoelmaan oma `IfcDataModel`-luokan dokumentti. Metadataa kuvaavat dokumentit tallennetaan IfcMetadata-kokoelmaan ja geometriaa kuvaavat dokumentit IfcGeometryData-kokoelmaan.

`IfcDataModel`-luokka sisältää attribuutit `RelatedProjectId`, `RelatedProjectFileId`, joiden avulla voidaan selvittää minkä projektin tietomallitiedostoon kyseinen metadata liittyy. Tällä vältytään referenssien ylläpitämiseltä Projects-kokoelmassa. Meta- ja geometriadatan versiointi toteutetaan `IfcData`-luokan avulla. `IfcData`-luokka sisältää `RelatedProjectFileVersion`-attribuutin, joka kertoo mihin tiedostoversioon luokan sisältämä data liittyy. Varsinainen IFC-tiedoston meta- ja geometriadata kuvataan luokan sisältämän `IfcObject`-attribuutin avulla, joka on `IFCEntity`-luokan olio.

`IFCEntity`-luokan avulla voidaan kuvata yksittäisen rakenneosan sisältämät metatiedot tai geometria. Tämä mahdollistaa sen, että, vaikka yksittäisen rakenneosan meta- ja geometriadata tallennetaan eri kokoelmiin, on se tarvittaessa yhdistettävissä. Tiedot voidaan yhdistää `IFCEntity`-luokan `GlobalId`-attribuutin avulla, jonka arvo saadaan IFC-tiedostosta. Muita meta- ja geometriadatalle yhteisiä attribuutteja ovat `Name`, `IfcType` ja `Description`, joihin tallennetaan rakenneosan nimi, rakenneosan tyyppi ja kuvaus.



Kuva 5.11 IFC-metadatan ja geometrian tietorakenne.

Metadatan kuvaamisessa käytetään hyväksi `IFCProperty`-luokkaa, jolla voidaan määrittää rakenneosan sisältämät ominaisuudet avain-arvo-pareina. Käytännössä `IFCProperty`-luokan sisältämät ominaisuudet ovat IFC-tiedostossa määriteltyjä `IfcPropertySetDefinition`-määritteitä. Tämän lisäksi `IFCEntity`-luokka voi sisältää listan `IFCEntity`-olioita,

jotka liittyvät siihen jollain tavalla. Näitä olioita ovat sellaiset IFC-tiedoston objektit, jotka eivät ole rakenneosia, kuten materiaalitiedot.

Rakenneosan geometria kuvataan `Geometry`-luokan avulla. Luokka sisältää attribuutit, joiden avulla rakenneosan geometria voidaan kuvata kolmioituna verkkona. Luokka sisältää attribuutit `positions`-, `triangleindices` ja `normals`, joiden avulla geometriaa kuvaava kolmioverkko määritellään. `Positions`-attribuutti pitää sisällään geometrian pisteet ja `triangleindices`-attribuutti pisteiden järjestyksen. `Normals`-attribuutti kertoo pinnan normaalin suunnan. Tietorakenne on hyvin lähellä jQuery-kirjaston Geometry-luokan tapaa kuvata geometriaa, joten se on helposti hyödynnettävissä tietomallin katselunäkymässä. Tämän lisäksi `Geometry`-luokka sisältää materiaalien värien kuvaamisen käytettävän `MaterialRGB`-luokan. Luokka pitää sisällään `R`-, `G`-, ja `B`-attribuutit, joiden avulla voidaan määritellä väri RGB-arvoina. Luokan sisältämää `A`-attribuuttia käytetään ilmaisemaan värin läpinäkyvyys. Tämän lisäksi luokalla on `Name`-atribuutti, johon tallennetaan IFC-tiedostossa olevan materiaalin nimi.

5.4 Käyttäjähallinta

Tietomallipalvelimen käyttäjähallinnassa hyödynnetään Microsoftin ASP.NET Identity-kirjastoa. Kirjasto on tarkoitettu käytettäväksi ASP.NET MVC- ja Web API-sovellusten käyttäjähallintaan. Kirjasto pitää sisällään valmiita luokkia, joiden avulla voidaan esittää käyttäjien tarvitsema tietosisältö sekä käyttäjien roolit. ASP.NET Identity tarjoaa myös luokat, joiden avulla käyttäjien tietoja ja käyttäjien rooleja voi hallita. (Introduction to ASP.NET Identity)

`IdentityUser`-luokan avulla voidaan esittää järjestelmän käyttäjien tarvitsema tietosisältö. Luokka pitää sisällään yleisimmin tarvitut käyttäjätiedot, kuten käyttäjätunnus, salasana, etu- ja sukunimi, sähköposti ja puhelinnumero. (Microsoft ASP.NET Identity Entity Framework) Käyttäjän tietokenttiä voi lisätä edelleen periyttämällä luokasta oman toteutuksensa. Tietomallipalvelimen toteutuksessa käyttäjille ei lisätty uusia tietokenttiä, vaan käyttäjiä kuvaamaan käytettiin `IdentityUser`-luokkaa.

Käyttäjien roolien avulla voidaan hallita sitä, mihin toimintoihin ja tietoihin käyttäjällä on oikeus päästä käsiksi. Roolien esittämiseen ASP.NET Identity tarjoaa `IdentityRole`-luokan. `IdentityRole`-luokan tietosisältö on yksinkertainen, ja se pitää sisällään vain roolin nimen sekä roolin yksittäisen id-tunnisteen. (Microsoft ASP.NET Identity Entity Framework) Tietomallipalvelimella oli tarvetta erottaa kolme eri käyttäjäroolia: ylläpitäjä, yrityksen sisäinen tai asiakas, joten sovellukselle lisättiin oletuksena nämä kolme roolia.

Toimintoihin ja tietoihin pääsyn logiikasta huolehtiminen jää sovelluskehittäjän vastuulle. Käytännössä oikeuksienhallinnan logiikka lisätään ASP.NET MVC ja ASP.NET Web API-sovelluksessa ohjain-luokkien sisältämien Action-metodien yhteyteen. Action-

metodeissa voidaan tarkistaa mihin rooliin metodin suoritusta pyytävä käyttäjä kuuluu ja sallia tai kieltää toiminnon suoritus roolin perusteella.

Käyttäjien ja roolien hallintaan ASP.NET Identity käyttää `UserManager`- ja `RoleManager`-luokkia. `UserManager`-luokan avulla voidaan luoda uusia käyttäjiä, muokata käyttäjien tietoja tai poistaa käyttäjiä. `UserManager` tarjoaa myös metodit, joilla käyttäjälle voi joko lisätä tai poistaa rooleja sekä selvittää mihin rooliin kyseinen käyttäjä kuuluu. `RoleManager`-luokan avulla voi luoda uusia rooleja, muokata rooleja tai poistaa niitä. (Microsoft ASP.NET Identity)

`UserManager`- ja `RoleManager`-luokat käyttävät tiedon tallentamisessa ja hakemisessa hyödyksi `UserStore`- ja `RoleStore`-luokkia. `UserStore`- ja `RoleStore`-luokat ovat yhteydessä tietokantaan. (Microsoft ASP.NET Identity)

5.5 Rajapinta

Tietomallipalvelimen rajapinta on toteutettu ASP.NET Web API-sovelluskehyksen avulla ja siinä on pyritty noudattamaan REST-arkkitehtuurityyliä. Rajapinta tarjoaa mahdollisuuden kolmannen osapuolen sovelluksille kommunikointiin tietomallipalvelimen kanssa.

Rajapinnan avulla voi hakea tietomallipalvelimella olevat projektit ja niihin liittyvät tiedostot. Rajapinnan kautta voi myös hakea IFC-tiedostosta parsittujen rakenneosien meta- ja geometriatietoja. Rajapintaan ei ole toteutettu tiedostojen lähetystoimintoa, koska sitä ei nähty tarpeelliseksi, sillä tiedostojen lähetys on mahdollista web-selaimella käytettävän käyttöliittymän kautta. Rajapinta on kuitenkin helposti laajennettavissa ja lisää toimintoja voidaan lisätä tulevaisuudessa tarpeen mukaan.

Rajapintaa käytetään HTTP-kutsujen avulla, joissa haluttu toiminto määritetään URI-osoitteen ja HTTP-metodin avulla. Palvelin palauttaa vastauksena, joko JSON- tai XML-muotoista dataa. Palautettava data vastaa projektien ja IFC-tiedoston meta- ja geometriatietojen osalta kohdassa 4.4 kuvattuja tietorakenteita. Osa palautettavasta datasta poikkeaa tästä hieman, koska sitä on optimoitu ajatellen tiedonsiirtoa ja pyritty poistamaan turhia tietoja. Esimerkiksi projektien listaus toiminto ei palauta kaikkia projektin sisältämien tiedostojen versioita ja metatietoja, vaan niistä palautetaan pelkästään tiedostojen lukumäärä. Tietomallipalvelimen rajapinnan tarkempi kuvaus on näkyvillä liitteessä C.

Rajapintaa hyödynnetään tietomallipalvelimen varmuuskopiointisovelluksessa, jota käytetään varmuuskopioiden palauttamiseen Tekla Structures-suunnittelusovellukseen. Rajapinnan tarjoamien toimintojen avulla sovellus voi palauttaa Tekla Structures-rakennemallin osille niiden metatietoja. Lisäksi sovelluksen avulla on mahdollista tarkastella miten rakenneosien metatiedot ovat muuttuneet verrattuna edelliseen versioon.

6. ARVIOINTI

Tässä luvussa arvioidaan, kuinka tietomallipalvelimen toteutuksessa on onnistuttu. Ensin esitellään tietomallipalvelimella tehty käyttökokeilu. Tämän jälkeen käydään läpi valittujen toteutustekniikoiden hyviä ja huonoja puolia. Lopuksi esitetään tietomallipalvelimen jatkokehitysajatuksia.

6.1 Käyttökokeilu

Tietomallipalvelinta on kokeiltu osana Android-mobiilisovellusta, joka kehitettiin Suomen ensimmäisessä AEC Hackathon-tapahtumassa Helsingissä vuonna 2015. AEC Hackathon on rakennusalan osaajille ja ohjelmistokehittäjille suunnattu tapahtuma, jossa osallistujat muodostavat keskenään joukkueita ja ratkaisevat alan haasteita yhden viikonlopun aikana. Kaiken kaikkiaan haasteisiin otti osaa 14 joukkuetta. Kyseisessä tapahtumassa haasteet oli jaettu kahdeksaan eri kategoriaan ja niiden aiheet tulivat tapahtuman yhteistyökumppaneilta, joita olivat esimerkiksi rakennusalan yritykset ja rakennusalaan liittyvät yhteisöt. (AEC Hackathon)

AEC Hackathonissa kehitetty mobiilisovellus osallistui tapahtumassa IndoorAtlas-yrityksen esittämään IndoorNav-kategoriaan, jossa oli tarkoitus hyödyntää yrityksen tarjoamaa sisätilapaikannukseen tarkoitettua ohjelmistokirjastoa ja luoda sen avulla hyödyllinen sovellus rakennusalalle. Ohjelmointikirjasto on saatavilla Android- ja iOS-alustoille (AEC Hackathon; IndoorAtlas developer section).

Tapahtumassa kehitetty mobiilisovellus näyttää käyttäjän sijainnin rakennuksen sisällä rakennuksen 3D-tietomallissa. Mobiilisovelluksen rakennuksena käytettiin AEC Hackathonin tapahtumapaikkaa. Mobiilisovellus sisältää myös navigointitoiminnon ja osaa ohjata käyttäjän haluamaansa sijaintiin. Kyseisessä sovelluksessa valittavissa olevina sijainteina käytettiin muutamaa ennalta määritettyä rakennuksessa olevaa kokoustilaa. Mobiilisovelluksen avulla voi lisäksi tarkastella rakennuksen rakenneosilla olevaa metatietoa, jota käyttäjä voi katsoa klikkaamalla sovelluksen näkymässä näkyvissä olevia rakenneosia.

Mobiilisovellus kehitettiin Unity-pelimoottorilla (http://unity3d.com/). Sovelluksessa näytettävä rakennuksen 3D-malli luotiin IFC-tietomallista, joka konvertoitiin Unity-pelimoottorille sopivaksi Blender-mallinnusohjelman kautta (https://www.blender.org/). Käyttäjän sijaintitiedon saamisessa käytetään IndoorAtlaksen ohjelmointikirjastoa, joka yhdistettiin Unity-pelimoottoriin.

Mobiilisovellus hyödyntää rakenneosien metatietojen näyttämisessä tietomallipalvelinta. Tätä varten rakennuksen 3D-mallin pohjana ollut IFC-tietomalli lähetettiin tietomallipalvelimelle. Tietomallipalvelin tarjoaa rajapinnan, josta mobiilisovellus hakee sen näkymässä näytettävät rakenneosien metatiedot.

AEC Hackathonissa kehitetty mobiilisovellus voitti tapahtuman parhaan projektin palkinnon (AEC Hackathon). Tapahtuman tuomaristo oli kansainvälinen ja koostui rakennus- ja ohjelmistoalan ammattilaisista. Tuomariston mielestä kyseinen sovellus vastasi parhaiten haasteeseen, jonka tarkoituksena oli ratkaista rakennusalan suurimpia haasteita. Voiton johdosta sovellus esiteltiin lisäksi Slush Hacks-finaalissa Helsingissä vuonna 2015.

Käyttökokeilua voidaan pitää onnistuneena, koska tietomallipalvelinta pystyttiin hyödyntämään osana mobiilisovellusta ja kokeilussa saatiin aikaan todistetusti rakennusalalle hyödyllinen sovellus. Kokeilulla osoitettiin, että tietomallipalvelimen sisältämää tietoa voidaan hyödyntää muuallakin, mikä laajentaa tietomallipalvelimen käyttömahdollisuuksia. Kokeilu tehtiin yhden viikonlopun aikana, joka osoittaa, että järjestelmä on suunniteltu niin, että sen integrointi muihin järjestelmiin voidaan tehdä nopealla aikataululla ja täten myös kustannustehokkaasti.

6.2 Kokemukset toteutustekniikoista

Tietomallipalvelimen toteutuksessa hyödynnetyt ASP.NET MVC ja ASP.NET Web API-sovelluskehykset osoittautuivat tehokkaiksi työkaluiksi käytettynä Microsoft Visual Studio-ohjelmointiympäristön kanssa. Microsoft Visual Studio tarjoaa monia hyviä työkaluja, joilla sovelluskehitystä on mahdollista nopeuttaa. Visual Studio pitää sisällään valmiit projektipohjat ASP.NET MVC- ja ASP.NET Web API-sovelluksille, jotka generoivat automaattisesti perussovelluksessa tarvittavat lähdekoodit. Visual Studio pitää sisällään myös IIS-palvelimen, jolla kehitettyä sovellusta voi testata paikallisesti ilman erillisiä asennuksia. Visual Studio mahdollistaa myös lähdekoodin debuggauksen samalla, kun sovellusta suoritetaan IIS-palvelimella.

Kehitystä nopeuttaa myös se, että Visual Studion avulla voi generoida lähdekoodia automaattisesti. Esimerkiksi malli-luokille voi luoda automaattisesti ohjain-luokan, joka sisältää perustoiminnot malli-luokan tietosisällön käsittelyyn. Lisäksi ohjain-luokille voi luoda automaattisesti perusnäkymät, joilla toimintoihin pääsee käsiksi.

Microsoft tarjoaa ohjesivuillaan paljon ohjemateriaalia ja esimerkkiprojekteja (Learn About ASP.NET MVC). Sivuilla neuvotaan, kuinka erilaiset perusominaisuudet olisi hyvä toteuttaa ja kuinka sovelluskehyksen kerrosarkkitehtuuria tulisi noudattaa. Tämä auttaa pääsemään alkuun sovelluskehityksessä ja oikeiden arkkitehtuuriratkaisujen suunnittelussa, vaikka sovelluskehys ei olisikaan tuttu entuudestaan.

Sovelluskehysten käytössä ilmeni kuitenkin myös ongelmia. Käytetyt ASP.NET MVC- ja ASP.NET Web API-sovelluskehykset eivät ole suoraan yhteensopivia, koska niiden eri osien komponetit, kuten ohjaimien toteutuksessa käytettävät luokat sijaitsevat eri nimiavaruuksissa. Tästä johtuen ohjain-luokat sisältävät päällekkäistä saman toiminnallisuuden toteuttavaa koodia. Tämä aiheuttaa haasteita siinä vaiheessa, kun toiminnallisuutta tai sovelluslogiikkaa tarvitsee muuttaa, koska muutoksia joutuu tekemään kahteen paikkaan. Sovelluskehyksiä voidaan kuitenkin käyttää rinnakkain, ja niissä voidaan hyödyntää esimerkiksi samoja malliluokkia.

ASP.NET MVC 6:ssa tämä ongelma kuitenkin poistuu, koska Web API on siinä yhdistetty samaan sovelluskehykseen (Migrating from ASP.NET Web API 2 to MVC 6). Tämä vähentää päällekkäistä koodia ja helpottaa sovelluslogiikan muuttamista. Tietomallipalvelimen toteutus olisikin tästä syystä järkevä muuntaa tulevaisuudessa käyttämään uutta sovelluskehystä.

IFC-tiedostojen parsintaan käytetty IFC Engine-kirjasto suoriutui useimpien tietomallipalvelimelle lähetettyjen IFC-tiedostojen käsittelystä. Joidenkin testattujen IFC-tiedostojen kanssa ilmeni ongelmia geometrian käsittelyssä. Ongelmat kuitenkin korjaantuivat vaihdettaessa kirjasto uudempaan versioon.

Kirjaston dokumentaatio ja ohjemateriaali on suppea, joten sovelluskehityksessä ei pääse nopeasti alkuun. Kirjaston tietorakenne perustuu IFC-standardiin, eikä abstrahoi tietorakennetta juurikaan. Tätä varten sovelluskehittäjä joutuu opiskelemaan myös IFC-standardia, jotta pystyy hyödyntämään kirjastoa tehokkaasti. Kehitystä hidasti myös se, että kirjasto on toteutettu C++-ohjelmointikielellä. Tätä varten kirjastolle oli luotava C#-ohjelmointikielellä wrapper-luokka, jotta sitä voitiin hyödyntää ASP.NET MVC- ja ASP.NET Web API-sovelluskehyksissä.

Tietomallipalvelimen 3D-näkymän toteutuksessa käytetty Three.js-kirjasto nopeutti sovelluskehitystä ja toteutus pelkän WebGL-teknologian avulla olisi vienyt huomattavasti enemmän aikaa. Kirjasto abstrahoi WebGL-teknologiaa ja tarjoaa monia valmiita komponentteja, kuten Geometry-luokan, jonka avulla avulla IFC-tietomallista parsittu geometria oli helposti esitettävissä. Lisäksi kirjasto ottaa huomioon eri selainten vaihtelevan WebGL-teknologia tuen ja osaa korjata tapaukset, joissa käytettävä näytönohjain ei tue WebGL:ää. Tästä on hyötyä kun halutaan luoda sovellus, joka toimii mahdollisimman monella eri laitteella.

Kirjasto ei kuitenkaan täysin poistanut WebGL-teknologian ongelmia. Eri selainten välillä on huomattavia suorituskykyeroja, jotka korostuvat kun tietomallien koko kasvaa. Suorituskykyä voidaan kuitenkin parantaa tulevaisuudessa luomalla tietomalleista eritarkkuustason versioita. Lisäksi suorituskykyä voidaan parantaa poistamalla 3D-näkymästä ne objektit, jotka ovat piilossa toisten objektien takana tai kameran näkökentän ulkopuolella.

Tietokannan toteutuksessa käytetty MongoDB mahdollisti IFC-tietomallien sisältämän tiedon tallentamisen joustavasti, koska tietokantaan tallennettavien dokumenttien skeemaa ei tarvitse määritellä tarkasti etukäteen. Jos tietokantana olisi käytetty normalisoitua relaatiotietokantaa, olisi luotavien taulujen määrä noussut suureksi. Tämä olisi monimutkaistanut toteutusta ja vaikuttanut myös suorituskykyyn.

MongoDB:n 16 megatavun maksimikoko yksittäiselle dokumentille asetti rajoitteita tietorakenteen suunnittelulle. Kokorajoitteesta johtuen tietomallin rakenneosia ja niiden eri versioita ei voitu tallentaa yksittäisenä dokumenttina. Tätä varten tietomallin jokaiselle rakenneosalle oli luotava oma dokumenttinsa. Tämä hidastaa hakuja hieman, koska useamman rakenneosan tietoja haettaessa on yhdistettävä useampia dokumentteja. Toisaalta suunnitteluratkaisu mahdollistaa nopeammat haut, jos halutaan hakea vain yhden rakenneosan tietoja. Yleisimmässä käyttötapauksessa 3D-malli ja sen kaikkien rakenneosien geometria haetaan palvelimelta vain kerran, jonka jälkeen käyttäjä suorittaa yksittäisten rakenneosien metatietojen hakuja.

6.3 Jatkokehitysajatuksia

Tällä hetkellä tietomallipalvelin mahdollistaa vain yhden tietomallin tarkastelun 3D-näkymässä. Tulevaisuudessa 3D-näkymässä olisi mahdollista näyttää useamman suunnittelualan malli kerralla. Yhdistämällä rakennesuunnittelu-, LVI- ja talotekniikka-mallit samaan näkymään voitaisiin tarkastella enemmän tietoa kerralla, joka tarjoaa uusia käyttömahdollisuuksia. Yhdistellyille malleille voitaisiin suorittaa esimerkiksi rakenneosien törmäystarkasteluja. Tällä voitaisiin auttaa eri suunnittelualojen tekemien suunnitelmien yhteensovituksessa ja korjata mahdollisia suunnitteluvirheitä.

Jos tietomalliin lisätään tietoa suunnittelun statuksesta, on tietomallipalvelimen avulla mahdollista nähdä suunnittelun tämän hetkinen tilanne. Tietomalliin voitaisiin lisätä tietoa myös suunnitellusta asennusaikataulusta. Tämän tiedon avulla myös esimerkiksi rakennuksen elementtien asennusaikataulua voitaisiin visualisoida 3D-mallin avulla. Tästä olisi hyötyä esimerkiksi rakennustyömaalla asennusjärjestyksen suunnittelussa ja hahmottamisessa.

Tämän lisäksi tietomallipalvelin voisi tarjota toimintoja, jonka avulla rakennustyömaalla voitaisiin merkitä tietomalliin asennuksen toteuma. Tällä tavoin eri rakennushankkeen osapuolet näkisivät rakennushankkeen tilanteen sekä myös mahdolliset poikkeamat verrattuna suunniteltuun aikatauluun.

Rakenneosille olisi myös mahdollista lisätä tietoa mahdollisista asennuksissa tapahtuneista poikkeamista verrattuna alkuperäiseen suunnitelmaan. Tämä tieto mahdollistaisi toteumamallin luonnin, joka on hyödynnettävissä rakennuksen ylläpitovaiheessa. Ylläpitovaihetta varten mallin osille voitaisiin linkittää esimerkiksi niihin liittyviä huoltokirjoja tai ohjedokumentteja.

Käyttökokeilussa tietomallipalvelinta hyödynnettiin IFC-mallin rakenneosien metatietojen hakemisessa. Jotta myös rakenneosien 3D-geometria olisi paremmin hyödynnettävissä ulkopuolisissa sovelluksissa, tulisi tietomallipalvelimen tarjota 3D-geometria myös jossain standardissa formaatissa, kuten X3D tai Collada (What is X3D; Collada). Tällöin sovelluskehittäjät voisivat hyödyntää 3D-geometrian esittämisessä jo olemassa olevia ohjelmointikirjastoja.

7. JOHTOPÄÄTÖKSET

Tässä diplomityössä toteutettiin rakennuksen 3D-tietomallipalvelin, johon voidaan tallentaa rakennushankkeessa suunniteltavien rakennusten tietomalleja ja niiden sisältämää geometria- ja metatietoa. Tietomallipalvelimen tarkoituksena on helpottaa eri rakennushankkeen osapuolten välistä tiedonsiirtoa. Tällä hetkellä tiedonsiirtoon käytetään projektipankkeja, jotka on suunniteltu dokumenttien siirtämiseen. Projektipankit eivät ymmärrä tietomallien sisältämää geometria- ja metatietoa. Tietomallipalvelin tarjoaa toiminnot, jonka avulla tietomallissa olevaa tietoa voidaan hakea ja päivittää. Tällöin rakennushankkeen osapuolet voivat hakea vain tarvitsemansa tiedot eikä heidän tarvitse ladata koko tietomallia.

Tietomallipalvelin toteutettiin A-Insinöörit Suunnittelu Oy:lle, jonka projektit ovat suurimmalta osin rakennesuunnittelua. A-Insinöörit Suunnittelu Oy:ssä koettiin, että yrityksessä on tarve omalle toteutukselle, koska markkinoilla olevat tietomallipalvelinten toteutukset keskittyvät arkkitehtisuunnitteluun ja eivät vastaa kaikkiin yrityksen tarpeisiin. Tästä johtuen tietomallipalvelimen toteutuksessa tuli ottaa huomioon rakennesuunnittelijan näkökulma. Myös tietomallipalvelimelle asetetut toiminnalliset vaatimukset kehitettiin A-Insinöörit Suunnittelu Oy:n rakennesuunnitteluprojekteissa ilmenneiden haasteiden ja ongelmien pohjalta.

Toinen tärkeä asia, joka loi tarpeen tietomallipalvelimen kehitykselle, on yrityksessä suunniteltujen Tekla Structures-suunnitteluosovelluksella suunniteltujen tietomallien varmuuskopioinnin parantaminen. Aiemmin tietomallien varmuuskopioinnin palautus piti tehdä koko tietomallille. Tietomallipalvelimen avulla varmuuskopioinnin palautus voidaan tarvittaessa tehdä vain valituille rakenneosille. Tästä johtuen tietomallipalvelimen piti tarjota rajapinta, jonka avulla myös kolmannen osapuolen sovellukset voivat käyttää tietomallipalvelimen tarjoamia toiminnallisuuksia.

Tietomallipalvelin toteutettiin web-sovelluksena, jota käytetään verkkoselaimella. Web-sovelluksen toteutuksessa käytettiin Microsoftin ASP .NET MVC-sovelluskehystä. Sovelluskehys valittiin, koska A-Insinöörit Oy:ssä on jo olemassa palvelinympäristö, joka tukee kyseisellä sovelluskehyksellä toteutettuja sovelluksia.

Toteutuksessa hyödynnettiin myös kolmannen osapuolen kirjastoja, joilla laajennettiin sovelluskehyksen toimintaa. Tietomallipalvelimella suoritettavien pitkäaikaisten tehtävien suorittamisessa käytettiin Hangfire-kirjastoa. Tietomallipalvelimelle lähetettyjen

IFC-tietomallien käsittelyssä käytettiin IFC Engine-kirjastoa. Tietomallipalvelimen tietosisällön ja tietomallien tallentamisessa käytettiin dokumenttipohjaista MongoDB-tietokantaa.

Tietomallipalvelimen rajapinta toteutettiin ASP.NET Web API-sovelluskehyksen avulla. Rajapinnan toteutuksessa pyrittiin noudattamaan REST-arkkitehtuurityyliä.

Tietomallipalvelimen toteutus sujui ilman suuria haasteita. Projektin alussa tehdyt teknologiavalinnat osoittautuivat toimiviksi. Sekä käytetty sovelluskehys, tietokanta että kolmannen osapuolen kirjastot saatiin toimimaan yhdessä. Eniten työtä aiheutti IFC:n käsittelyyn käytetty IFC Engine-kirjasto, jota jouduttiin hyödyntämään erilliseen C# Wrapper-luokan kautta.

Työn tuloksena saatiin aikaan tietomallipalvelin, joka toteuttaa työlle asetetut tavoitteet. Tietomallipalvelimelle voi tallentaa IFC-tietomalleja ja tietomallipalvelin tarjoaa toiminnot, jonka kautta käyttäjät voivat hakea IFC-tietomallien sisältämiä geometria- ja metatietoja. Tietojen tarkasteluun tietomallipalvelin tarjoaa 3D-katselunäkymän, jossa näytetään IFC-tietomallin sisältämien rakenneosien geometria 3D-mallina sekä rakenneosiin liittyviä metatietoja. Tietomallipalvelin tarjoaa myös rajapinnan, jonka kautta kolmannen osapuolen sovellukset voivat hakea tietoa tietomallipalvelimelle tallennetuista tietomalleista. Rajapintaa hyödynnetään tietomallien varmuuskopioinnin palautukseen toteutetussa sovelluksessa.

Tietomallipalvelimen käyttökokeilu osoitti, että järjestelmä on myös integroitavissa tarvittaessa muihinkin järjestelmiin. Tämä luo paljon uusia mahdollisuuksia ja käyttötapauksia sille, missä tietomallipalvelinta voi hyödyntää.

LÄHTEET

AEC Hackathon, 2015, [WWW], [Viitattu 2.12.2015]. Saatavissa: http://aechackathon.com/aec-hackathon-2-7-helsinki/

ASP.NET MVC Overview, 2009, [WWW]. [Viitattu 28.1.2015]. Saatavissa: http://www.asp.net/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

BIMserver About, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://bimserver.org/about/who-is-doing-this/

BIMserver License, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://bimserver.org/license/

BIMserver Wiki, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: https://github.com/opensourceBIM/BIMserver/wiki

BIMsurfer, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://bimsurfer.org/

buildingSMART Current projects, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://www.buildingsmart-tech.org/future-extensions/ifc_extension_projects/current

Beetz, J., van Berlo, L., de Laat, R., van den Helm, P., 2010, Bimserver.org - An open source ifc model server, [WWW]. [Viitattu 8.1.2015]. Saatavissa: http://bimserver.org/wp-content/uploads/sites/6/2010/11/Beetz_Berlo-CIB-W78_Cairo.pdf

Collada, 2015, [WWW], [Viitattu 2.12.2015]. Saatavissa: https://www.khronos.org/collada/

Danchilla, B., 2012, Beginning WebGL for HTML5 [verkkokirja]. [Viitattu 19.4.2015]. Saatavissa rajoitetusti: http://library.books24x7.com/toc.aspx?bkid=54070

BSON Specification, 2015, [WWW]. [Viitattu 23.3.2015]. Saatavissa: http://bsonspec.org

Ciliberti, J., 2013, ASP.NET MVC 4 Recipes: A Problem-Solution Approach [verkkokirja]. [Viitattu 28.1.2015]. Saatavissa rajoitetusti: http://library.books24x7.com/toc.aspx?bkid=52966

Eastman, C., Teicholz, P., Sacks, R., Liston, K., 2011, BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors, Second Edition [verkkokirja]. [Viitattu 18.12.2014]. Saatavissa rajoitetusti: http://library.books24x7.com/toc.aspx?bkid=41695

ECMA-334, 2006, [WWW], [Viitattu 5.4.2015]. Saatavissa: http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf

Extensible Markup Language, 2008, [WWW], [Viitattu 2.12.2015]. Saatavissa: http://www.w3.org/TR/REC-xml/

Fielding, R., 2000, Architectural Styles and the Design of Network-based Software Architectures [väitöskirja], University of California, Irvine, Viitattu [3.5.2015]. Saatavissa: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Freeman, A., 2014a. Expert ASP.NET Web API 2 for MVC Developers [verkkokirja]. Viitattu [15.3.2015] Saatavissa rajoitetusti: http://library.books24x7.com/toc.aspx?bkid=70723

Freeman, A., 2014b, Pro ASP.NET MVC 5 Platform [verkkokirja]. Viitattu [28.1.2015]. Saatavissa rajoitetusti: http://library.books24x7.com/toc.aspx?bkid=65477

Hangfire, 2015, [WWW], [Viitattu 31.3.2015]. Saatavissa: http://hangfire.io

Hangfire Overview, 2015, [WWW], [Viitattu 31.3.2015]. Saatavissa: http://hangfire.io/overview.html

Hietanen, J., 2005. Tietomallit ja rakennusten suunnittelu – Filosofinen selvitys tieto- ja viestintätekniikan mahdollisuuksista, Rakennustieto Oy, Helsinki, 95 s.

Hows, D., Membrey, P., Plugge, E., Hawkins, T., 2013, The Definive Guide to MongoDB: A Complete Guide to Dealing with Big Data using MongoDB [verkkokirja]. Viitattu [28.1.2015]. Saatavissa rajoitetusti: http://library.books24x7.com/toc.aspx?bkid=59087

IFC2x3 TC1 Specification, 2007, [WWW]. [Viitattu 26.1.2015]. Saatavissa: http://www.buildingsmart-tech.org/ifc/IFC2x3/TC1/html/index.htm

IFC4 Specification, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://www.buildingsmart-tech.org/ifc/IFC4/final/html/index.htm

IFC Engine, 2015, [WWW]. [Viitattu 1.4.2015]. Saatavissa: http://rdf.bg/ifc-engine-dll.php

IFC Overview summary, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://www.buildingsmart-tech.org/specifications/ifc-overview

IndoorAtlas developer section, 2015, [WWW]. [Viitattu 2.12.2015]. Saatavissa: http://docs.indooratlas.com/

Internet Engineering Task Force (IETF), Hypertext Transfer Protocol: Semantics and Content, 2014a, [WWW]. [Viitattu 4.5.2015]. Saatavissa: http://tools.ietf.org/html/rfc7231

Internet Engineering Task Force (IETF), Hypertext Transfer Protocol: Caching, 2014b, [WWW]. [Viitattu 4.5.2015]. Saatavissa: http://tools.ietf.org/html/rfc7234

Introduction to ASP.NET Identity, 2013, [WWW], [Viitattu 2.12.2015]. Saatavissa: http://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity

Introduction to LINQ Queries, 2013, [WWW]. [Viitattu 1.4.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/bb397906.aspx

ISO/IEC 23270, 2006, [WWW]. [Viitattu 4.1.2015]. Saatavissa rajoitetusti: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=42926

Introduction to MongoDB, 2015, [WWW], [Viitattu 23.3.2015]. Saatavissa: http://docs.mongodb.org/manual/core/introduction/

JavaScript Object Notation, 2006, [WWW], [Viitattu 2.12.2015]. Saatavissa: http://www.ietf.org/rfc/rfc4627.txt

Koskimies, K., Mikkonen, T., 2005. Ohjelmistoarkkitehtuurit, Talentum oyj, Jyväskylä, 250 s.

Lea, R., Matsuda, K., 2013, WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL, Pearson Education Inc., Michigan, 600 p.

Learn About ASP.NET MVC, 2015, [WWW], [Viitattu 2.12.2015]. Saatavissa: http://www.asp.net/mvc

McGrath, M., 2014, jQuery [verkkokirja]. Viitattu [3.6.2015]. Saatavissa rajoitetusti: http://library.books24x7.com/toc.aspx?bkid=74486

Microsoft ASP.NET Identity, 2015, [WWW]. [Viitattu 2.12.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/microsoft.aspnet.identity(v=vs.108).aspx

Microsoft ASP.NET Identity Entity Framework, 2015, [WWW]. [Viitattu 2.12.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/microsoft.aspnet.identity.entityframework(v=vs.108).aspx

Migrating from ASP.NET Web API 2 to MVC 6, 2015, [WWW], [Viitattu 2.12.2015]. Saatavissa: http://docs.asp.net/projects/mvc/en/latest/migration/migrating-fromwebapi2.html

MongoDB Manual, 2015, [WWW]. [Viitattu 23.2.2015]. Saatavissa: http://docs.mongodb.org/manual/

Network Working Group (NWG), 2005, Uniform Resource Identifier: Generic Syntax [WWW]. [Viitattu 4.5.2015]. Saatavissa: http://tools.ietf.org/html/rfc3986

Open GL, 2015, [WWW], [Viitattu 4.5.2015]. Saatavissa: https://www.opengl.org/

Open GL ES, 2015, [WWW], [Viitattu 4.5.2015]. Saatavissa: https://www.khronos.org/opengles/

Summary of IFC Releases, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://www.buildingsmart-tech.org/specifications/ifc-releases

SOAP, 2007, [WWW]. [Viitattu 2.12.2015]. Saatavissa: http://www.w3.org/TR/soap12/

Tekla Field 3D, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://www.teklabimsight.com/tekla-field-3d

Tekla Field 3D Features, 2015, [WWW]. [Viitattu 4.1.2015]. Saatavissa: http://www.teklabimsight.com/tekla-field3d-features

Tekla Open API Developer's Guide, 2012, [WWW]. [Viitattu 20.1.2015]. Saatavissa rajoitetusti: https://extranet.tekla.com/BC/tekla-structures-en/developers/openapi/Documents/webhelp-181/Default.htm

Tekla Structures-suunnitteluohjelmisto, 2015, [WWW]. [Viitattu 4.12.2014]. Saatavissa: http://www.tekla.com/fi/tuotteet/tekla-structures

The OpenGL ES Shading Language, 2009, [WWW]. [Viitattu 4.5.2015]. Saatavissa: https://www.khronos.org/files/opengles_shading_language.pdf

Three.js, 2015, [WWW]. [Viitattu 2.4.2014]. Saatavissa: http://threejs.org/

Three.js docs, 2015, [WWW]. [Viitattu 22.3.2014]. Saatavissa: http://threejs.org/docs/

Ugurlu, T., Zeitler, A., Kheyrollahi, A., 2013, Pro ASP.NET Web API: HTTP Web Services in ASP.NET [verkkokirja]. [Viitattu 15.3.2015]. Saatavissa rajoitetusti: http://library.books24x7.com/toc.aspx?bkid=58397

Visual Studio, 2015, [WWW]. [Viitattu 2.4.2015]. Saatavissa: https://www.visualstudio.com/

Visual Studio Languages, 2015, [WWW]. [Viitattu 1.4.2015]. Saatavissa: https://msdn.microsoft.com/en-us/library/vstudio/ee822860(v=vs.100).aspx

WebGL Specification, 2014, [WWW]. [Viitattu 10.5.2015]. Saatavissa: https://www.khronos.org/registry/webgl/specs/1.0/

WebGL Public Wiki, 2015, [WWW]. [Viitattu 10.5.2015]. Saatavissa: https://www.khronos.org/webgl/

What is X3D, 2015, [WWW], [Viitattu 2.12.2015]. Saatavissa: http://www.web3d.org/x3d/what-x3d

LIITE A. IFC ENGINE C# WRAPPER-LUOKKA

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   using System.Runtime.InteropServices;
8
9   namespace IFCEngineModule
10  {
11      public class IFCEngineWrapper
12      {
13          public static readonly Int64 flagbit2 = 4;          // 2^^2    0000.0000..0000.0100
14          public static readonly Int64 flagbit3 = 8;          // 2^^3    0000.0000..0000.1000
15          public static readonly Int64 flagbit5 = 32;         // 2^^5    0000.0000..0010.0000
16          public static readonly Int64 flagbit8 = 256;        // 2^^8    0000.0001..0000.0000
17          public static readonly Int64 flagbit12 = 4096;      // 2^^12   0001.0000..0000.0000
18
19          public static readonly Int64 sdaiADB = 1;
20          public static readonly Int64 sdaiAGGR = sdaiADB + 1;
21          public static readonly Int64 sdaiBINARY = sdaiAGGR + 1;
22          public static readonly Int64 sdaiBOOLEAN = sdaiBINARY + 1;
23          public static readonly Int64 sdaiENUM = sdaiBOOLEAN + 1;
24          public static readonly Int64 sdaiINSTANCE = sdaiENUM + 1;
25          public static readonly Int64 sdaiINTEGER = sdaiINSTANCE + 1;
26          public static readonly Int64 sdaiLOGICAL = sdaiINTEGER + 1;
27          public static readonly Int64 sdaiREAL = sdaiLOGICAL + 1;
28          public static readonly Int64 sdaiSTRING = sdaiREAL + 1;
29          public static readonly Int64 sdaiUNICODE = sdaiSTRING + 1;
30          public static readonly Int64 sdaiEXPRESSSTRING = sdaiUNICODE + 1;
31          public static readonly Int64 engiGLOBALID = sdaiEXPRESSSTRING + 1;
32
33          public const string IFCEngineDLL = "ifcengine.dll";
34
35          [DllImport(IFCEngineDLL, EntryPoint = "setStringUnicode")]
36          public static extern Int64 setStringUnicode(bool unicode);
37
38          [DllImport(IFCEngineDLL, EntryPoint = "sdaiOpenModelBN")]
39          public static extern Int64 sdaiOpenModelBN(Int64 repository,
40                                                     string fileName,
41                                                     string schemaName);
42
43          [DllImport(IFCEngineDLL, EntryPoint = "sdaiSaveModelBN")]
44          public static extern void sdaiSaveModelBN(Int64 repository, string fileName);
45
46          [DllImport(IFCEngineDLL, EntryPoint = "GetSPFFHeaderItem", CharSet = CharSet.None)]
47          public static extern Int64 GetSPFFHeaderItem(Int64 model,
48                                                       Int64 itemIndex,
49                                                       Int64 itemSubIndex,
50                                                       Int64 valueType,
51                                                       out IntPtr outputValue);
52
53          [DllImport(IFCEngineDLL, EntryPoint = "sdaiGetEntityExtentBN")]
54          public static extern Int64 sdaiGetEntityExtentBN(Int64 model, string entityName);
55
56          [DllImport(IFCEngineDLL, EntryPoint = "sdaiGetMemberCount")]
57          public static extern Int64 sdaiGetMemberCount(Int64 aggregate);
58
59          [DllImport(IFCEngineDLL, EntryPoint = "sdaiGetAttrBN", CharSet = CharSet.None)]
60          public static extern Int64 sdaiGetAttrBN(Int64 instance, string attributeName,
61                                                   Int64 valueType,
62                                                   out IntPtr outputValue);
63
64          [DllImport(IFCEngineDLL, EntryPoint = "sdaiGetEntity", CharSet = CharSet.None)]
65          public static extern Int64 sdaiGetEntity(Int64 model, string entityName);
66
67          [DllImport(IFCEngineDLL, EntryPoint = "engiGetInstanceClassInfo", CharSet = Char-
68  Set.None)]
69          public static extern IntPtr engiGetInstanceClassInfo(Int64 entity);
70
```

```
71          [DllImport(IFCEngineDLL, EntryPoint = "engiGetInstanceLocalId", CharSet = CharSet.None)]
72          public static extern IntPtr engiGetInstanceLocalId(Int64 instance);
73
74          [DllImport(IFCEngineDLL, EntryPoint = "engiGetInstanceClassInfoUC", CharSet = Char-
75  Set.None)]
76          public static extern IntPtr engiGetInstanceClassInfoUC(Int64 instance);
77
78          [DllImport(IFCEngineDLL, EntryPoint = "engiGetInstanceMetaInfo", CharSet = Char-
79  Set.None)]
80          public static extern IntPtr engiGetInstanceMetaInfo(Int64 instance,
81                                                              Int64 localId,
82                                                              string className,
83                                                              string classNameUC);
84
85          [DllImport(IFCEngineDLL, EntryPoint = "engiGetAggrElement")]
86          public static extern Int64 engiGetAggrElement(Int64 aggregate,
87                                                        Int64 elementIndex,
88                                                        Int64 valueType,
89                                                        ref Int64 pValue);
90
91          [DllImport(IFCEngineDLL, EntryPoint = "initializeModellingInstance")]
92          public static extern Int64 initializeModellingInstance(Int64 model,
93                                                                 ref Int64 noVertices,
94                                                                 ref Int64 noIndices,
95                                                                 double scale,
96                                                                 Int64 instance);
97
98          [DllImport(IFCEngineDLL, EntryPoint = "initializeModellingInstanceEx")]
99          public static extern Int64 initializeModellingInstanceEx(Int64 model,
100                                                                ref Int64 noVertices,
101                                                                ref Int64 noIndices,
102                                                                double scale,
103                                                                Int64 instance,
104                                                                Int64[] instanceList);
105
106         [DllImport(IFCEngineDLL, EntryPoint = "finalizeModelling")]
107         public static extern Int64 finalizeModelling(Int64 model, float[] pVertices,
108                                                      Int64[] pIndices,
109                                                      Int64 FVF);
110
111         [DllImport(IFCEngineDLL, EntryPoint = "setFormat")]
112         public static extern void setFormat(Int64 models, Int64 setting,
113                                             Int64 masks);
114
115         [DllImport(IFCEngineDLL, EntryPoint = "sdaiCloseModel")]
116         public static extern void sdaiCloseModel(Int64 model);
117
118         [DllImport(IFCEngineDLL, EntryPoint = "getConceptualFaceCnt")]
119         public static extern Int64 getConceptualFaceCnt(Int64 instance);
120
121         [DllImport(IFCEngineDLL, EntryPoint = "sdaiGetInstanceType")]
122         public static extern Int64 sdaiGetInstanceType(Int64 instance);
123
124         [DllImport(IFCEngineDLL, EntryPoint = "getConceptualFaceEx")]
125         public static extern void getConceptualFaceEx(Int64 instance,
126                                                       Int64 index,
127                                                       out Int64 startIndexTriangles,
128                                                       out Int64 noIndicesTriangles,
129                                                       out Int64 startIndexLines,
130                                                       out Int64 noIndicesLines,
131                                                       out Int64 startIndexPoints,
132                                                       out Int64 noIndicesPoints,
133                                                       out Int64 startIndexFacesPolygons,
134                                                       out Int64 noIndicesFacesPolygons,
135                                                       out Int64 startIndexConceptualFacePolygons,
136                                                       out Int64 noIndicesConceptualFacePolygons);
137
138         [DllImport(IFCEngineDLL, EntryPoint = "owlGetInstance")]
139         public static extern void owlGetInstance(Int64 model,
140                                                  Int64 instance,
141                                                  ref Int64 owlInstance);
142
143         [DllImport(IFCEngineDLL, EntryPoint = "cleanMemory")]
144         public static extern void cleanMemory(Int64 model, Int64 mode);
```

```
145
146             [DllImport(IFCEngineDLL, EntryPoint = "GetVolume")]
147             public static extern double GetVolume(Int64 owlInstance,
148                                                    float[] pVertices,
149                                                    int[] pIndices);
150
151             [DllImport(IFCEngineDLL, EntryPoint = "GetArea")]
152             public static extern double GetArea(Int64 owlInstance,
153                                                 float[] pVertices,
154                                                 int[] pIndices);
155
156             [DllImport(IFCEngineDLL, EntryPoint = "GetPerimeter")]
157             public static extern double GetPerimeter(Int64 owlInstance);
158         }
```

LIITE B. WEBGL-ESIMERKKISOVELLUS

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
    <meta charset="utf-8" />
    <title>WebGL test</title>
</head>
<body onload="main()">
    <canvas id="webgl" width="400" height="400"></canvas>

    <!-- Vertex-shader -->
    <script id="shader-vs" type="x-shader/x-vertex">
        attribute vec3 aVertexPosition;
        void main() {
            gl_Position = vec4(aVertexPosition, 1.0);
        }
    </script>

    <!-- Fragment-shader -->
    <script id="shader-fs" type="x-shader/x-fragment">
        void main() {
            gl_FragColor = vec4(1.0,0.0,0.0, 1.0);
        }
    </script>

    <script>
        function main() {
            // Haetaan canvas-elementti id:n perusteella HTML-dokumentista,
            // johon piirtäminen suoritetaan.
            var canvas = document.getElementById("webgl");
            var gl = null;

            try {
                // Haetaan webgl-konteksti gl-muuttujaan.
                gl = canvas.getContext("webgl") || canvas.getContext("experi-
mental-webgl");
            }
            catch (e) { }

            if (!gl)
                return;

            // Ladataan shaderit
            var glProgram = loadShaders(gl);

            // Asetetaan geometria
            setupGeometry(gl);

            // Tyhjennetään canvas
            clearCanvas(gl);

            // Piirretään canvakselle
            draw(gl, glProgram);
        }

        function clearCanvas(gl) {
            // Asetetaan tyhjä väri
            gl.clearColor(0.0, 0.0, 0.0, 1.0);
            // Tyhjennetään canvas
            gl.clear(gl.COLOR_BUFFER_BIT);
```

```
60          }
61
62          function loadShaders(gl) {
63              // Luodaan shader-oliot
64              var vertexShader = gl.createShader(gl.VERTEX_SHADER);
65              var fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
66
67              // Haetaan GLSL ES-koodi HTML-dokumentin script-elementeiltä id:n pe-
68  rusteella
69              var vertexShaderSource = document.getElementById('shader-vs').in-
70  nerHTML;
71              var fragmentShaderSource = document.getElementById('shader-fs').in-
72  nerHTML;
73
74              // Asetetaan shadereiden lähdekoodi
75              gl.shaderSource(vertexShader, vertexShaderSource);
76              gl.shaderSource(fragmentShader, fragmentShaderSource);
77
78              // Käännetään shadereiden lähdekoodi
79              gl.compileShader(vertexShader);
80              gl.compileShader(fragmentShader);
81
82              // Luodaan sovellus
83              var glProgram = gl.createProgram();
84              // Liitetään shaderit sovellukseen
85              gl.attachShader(glProgram, vertexShader);
86              gl.attachShader(glProgram, fragmentShader);
87              // Linkataan sovellus
88              gl.linkProgram(glProgram);
89
90              // Tarkistetaan onnistuko linkkaus
91              if (!gl.getProgramParameter(glProgram, gl.LINK_STATUS)) {
92                  // Virhe
93              }
94
95              // Käsketään webgl-kontekstin käyttää sovellusta.
96              gl.useProgram(glProgram);
97
98              return glProgram;
99          }
100
101         function setupGeometry(gl) {
102             // Määritetään geometrian pisteet (x,y,z)-koordinaateilla.
103             var vertices = [
104                 -0.5, -0.5, 0.0,
105                 0.0, 0.5, 0.0,
106                 0.5, -0.5, 0.0
107             ];
108
109             // Luodaan bufferi, johon geometrian pisteet asetetaan.
110             var verticeBuffer = gl.createBuffer();
111             gl.bindBuffer(gl.ARRAY_BUFFER, verticeBuffer);
112             gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices),
113 gl.STATIC_DRAW);
114         }
115
116         function draw(gl, glProgram) {
117             // Sidotaan vertex-shader aVertexPosition-attribuutti geometrian
118 määrittelevään bufferiin.
119             var vertexPositionAttribute = gl.getAttribLocation(glProgram,
120 "aVertexPosition");
121             gl.enableVertexAttribArray(vertexPositionAttribute);
```

```
122                 gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false,
123     0, 0);
124 
125                 // Piirretään geometria canvas-elementille
126                 gl.drawArrays(gl.TRIANGLES, 0, 3);
127             }
128         </script>
129     </body>
130     </html
```

LIITE C. TIETOMALLIPALVELIMEN RAJAPINTADOKUMENTAATIO

KIRJAUTUMINEN

Kirjaa käyttäjän sisälle palvelimelle ja palauttaa access_token-parametrin, jota tulee käyttää toiminnoissa, jotka vaativat kirjautumista.	
URI ja parametrit	/Token
HTTP-metodi	POST
HTTP-pyynnön sisältö	grant_type=password&username={kayttajatunnus}&password={salasana} grant_type: password (Kiinteä valinta salasanatyypille). username: Käyttäjätunnus. password: Käyttäjän salasana.
Vastauksen HTTP-statuskoodi	Onnistunut pyyntö: 200 OK

Http-vastaus	{
	access_token:
	"FBexBGU8SdRSFL64EIqjmfNSo1929dT_gUjvFC1pRD0Ead94P7Qs_N793Fyz1R3mPHFZH5MkLVohjfXq8wVz1eFdHl-20wWTnvDs5g9HRIo3_LvQpGn1aEJRpUBJwbcfTMg3nX2-N7yxtk_FhzLu6_BLv_crz8aOaBpYtzM3My9jaiijS2qHFD6Mu65JWz7i2cv9XLIupqvC-aoDkay-fPD6E5GLo7eNqxN-mTjie_6aQqmzYd_aqrP8hPmDtaewU-7TnUDhWKJpemtmo_eo2WGEZZ4q_WOG-Rwp6pAgjY-1gK432Fpl473ZNU3xmcmAIdfu7N-RkqQS9CqroPsifQ6RyU-wXQShF-C225KToo-VOKzO6_wZfOQ8cSKwFDa-4TuOfJceKS-69AzAtfzpwY2hfRxrx-oboVzWchHhN95ljCFGFQhQhsX3yv_GNUnXvMU1DrG7Z_8_YRPGH2uOzUJSaHOU0lVs1ezSPZ_RgUz-VCamHPkHdN3_SoUQxyPO3jHtWov0J9vhjaxA7HeAwRv9Gw"
	token_type: "bearer"
	expires_in: 1209599
	userName: "jani.mt.nevalainen@gmail.com"
	.issued: "Fri, 04 Dec 2015 11:31:15 GMT"
	.expires: "Fri, 18 Dec 2015 11:31:15 GMT"
	}

PROJEKTIEN LISTAUS

Palauttaa listan tietomallipalvelimella olevista projekteista.	
URI	/api/projects
HTTP-metodi	GET
Http-pyynnön Otsikkotiedot	Authorization: Bearer {Token} Token: Access_token-parametrin arvo, joka saadaan kirjautumisen yhteydessä.
Parametrit	
Vastauksen HTTP-statuskoodi	Onnistunut pyyntö: 200 OK
Http-vastaus	[{ "Id" : "B874B663-083F-40B0-9998-481E596D77C1", "Name" : "Testiprojekti", "CreatorId" : "D234B663-083F-40B0-9998-481E596D77C0", "CreationTime" : "2015-11-26T10:35:16.3588819+02:00", "LastModified" : "2015-11-26T10:35:16.35988+02:00", "Description" : "Projekti tehty testausta varten", "ProjectFiles" : 1, "URL" : "/api/projects/B874B663-083F-40B0-9998-481E596D77C1"

	}, {
	"Id" : "B874B663-083F-40B0-9998-481E596D77C1",
	"Name" : "Testiprojekti 2",
	"CreatorId" : "D234B663-083F-40B0-9998-481E596D77C0",
	"CreationTime" : "2015-11-26T10:35:16.3608791+02:00",
	"LastModified" : "2015-11-26T10:35:16.3608791+02:00",
	"Description" : "Projekti tehty testausta varten",
	"ProjectFiles" : 1,
	"URL" : "/api/projects/B874B663-083F-40B0-9998-481E596D77C1"
	}
	]

YKSITTÄISEN PROJEKTIN TIEDOT

Palauttaa yksittäisen projektin tiedot ja siihen liittyvien tiedostojen metatiedot.	
URI ja parametrit	/api/projects/{id} id: Projektin id.
HTTP-metodi	GET
Http-pyynnön Otsikkotiedot	Authorization: Bearer {Token} Token: Access_token-parametrin arvo, joka saadaan kirjautumisen yhteydessä.
Vastauksen HTTP-status-koodi	Onnistunut pyyntö: 200 OK

Http-vastaus	{
	"Id" : "B874B663-083F-40B0-9998-481E596D77C1",
	"Name" : "Testiprojekti",
	"CreatorId" : "D234B663-083F-40B0-9998-481E596D77C0",
	"CreationTime" : "2015-11-26T10:58:21.7176496+02:00",
	"LastModified" : "2015-11-26T10:58:21.7176496+02:00",
	"Description" : "Projekti tehty testausta varten",
	"ProjectFiles" : [{
	"Id" : "A874B663-083F-40B0-9998-481E596D2340",
	"Title" : "Testitiedosto",
	"Description" : "",
	"CreationTime" : "2015-11-26T10:58:21.7176496+02:00",
	"LastModified" : "0001-01-01T00:00:00",
	"FileVersions" : [{
	"FileName" : "tiedosto.ifc",
	"Type" : null,
	"Author" : "A874B663-083F-40B0-9998-481E596D2340",
	"FileLength" : 123123,
	"State" : 0,
	"CreationTime" : "2015-11-26T10:58:21.7176496+02:00",
	"LastModified" : "0001-01-01T00:00:00",
	"Version" : 1,
	"URL" : "/api/projects/B874B663-083F-40B0-9998-481E596D77C1/files/A874B663-083F-40B0-9998-481E596D2340/versions/1"
	}
	]
	}
	]
	}

PROJEKTIN TIEDOSTOVERSIOON LIITTYVÄT LISÄTIEDOT

Palauttaa projektin tiedostoversion linkit, joilla voi hakea alkuperäisen IFC-tiedoston ja siitä parsitut meta- ja geometriatiedot.	
URI ja parametrit	/api/projects/{id}/files/{fileId}/versions/{version} id: Projektin id. fileId: Tiedoston id.

	version: Tiedoston versionumero.
HTTP-metodi	GET
Http-pyynnön Otsikkotiedot	Authorization: Bearer {Token} Token: Access_token-parametrin arvo, joka saadaan kirjautumisen yhteydessä.
Vastauksen HTTP-status-koodi	Onnistunut pyyntö: 200 OK

Http-vastaus	[{
	"DataType" : "file",
	"URL" : "api/projects/B874B663-083F-40B0-9998-481E596D77C1/files/A874B663-083F-40B0-9998-481E596D2340/versions/1/file"
	}, {
	"DataType" : "metadata",
	"URL" : "api/projects/B874B663-083F-40B0-9998-481E596D77C1/files/A874B663-083F-40B0-9998-481E596D2340/versions/1/metada-tas"
	}, {
	"DataType" : "geometry",
	"URL" : "api/projects/B874B663-083F-40B0-9998-481E596D77C1/files/A874B663-083F-40B0-9998-481E596D2340/versions/1/geometry-datas"
	}
	]

PROJEKTIN TIEDOSTON LATAUS

Palauttaa alkuperäisen tietomallipalvelimelle ladatun IFC-tiedoston.	
URI ja para-metrit	/api/projects/{id}/files/{fileId}/versions/{version}/file id: Projektin id. fileId: Tiedoston id. version: Tiedoston versionumero.
HTTP-metodi	GET
Http-pyynnön Otsikkotiedot	Authorization: Bearer {Token} Token: Access_token-parametrin arvo, joka saadaan kirjautumisen yhteydessä.
Vastauksen HTTP-status-koodi	Onnistunut pyyntö: 200 OK

Http-vastaus	[{
	"DataType" : "file",
	"URL" : "api/projects/B874B663-083F-40B0-9998-481E596D77C1/files/A874B663-083F-40B0-9998-481E596D2340/versions/1/file"
	}, {
	"DataType" : "metadata",
	"URL" : "api/projects/B874B663-083F-40B0-9998-481E596D77C1/files/A874B663-083F-40B0-9998-481E596D2340/versions/1/metada-tas"
	}, {
	"DataType" : "geometry",
	"URL" : "api/projects/B874B663-083F-40B0-9998-481E596D77C1/files/A874B663-083F-40B0-9998-481E596D2340/versions/1/geometry-datas"
	}
	]

PROJEKTIN TIEDOSTON METATIETOJEN HAKU

Palauttaa projektin tiedostosta parsitut metatiedot.	
URI ja para-metrit	/api/projects/{id}/files/{fileId}/versions/{version}/metadatas id: Projektin id.

	fileId: Tiedoston id. verison: Tiedoston versionumero.
HTTP-metodi	GET
Http-pyynnön Otsikkotiedot	Authorization: Bearer {Token} Token: Access_token-parametrin arvo, joka saadaan kirjautumisen yhteydessä.
Vastauksen HTTP-status-koodi	Onnistunut pyyntö: 200 OK

Http-vastaus	{
	"Id" : 562236f60bc64c1978c07d34,
	"RelatedProjectId" : 562236f60bc64c1978c07d35,
	"RelatedProjectFileId" : 562257700bc65d197803a59a,
	"IfcObjects" :
	[{
	"Ifctype" : "IfcColumn",
	"GlobalId" : "1M2ceu0029tZ4qD3CrCZGp",
	"Name" : "COLUMN",
	"Description" : "HEA400",
	"FatherGlobalId" : "0WCujW4VT8Mg6cuJAkxH8Z",
	"Children" : [{
	"Ifctype" : "IfcMaterial",
	"Name" : "STEEL/S235JR",
	"Children" : null,
	"Properties" : []
	}
	],
	"Properties" : [{
	"Property" : "Weight",
	"Value" : "961.100000",
	"PropertyType" : "IFCMASSMEASURE"
	}, {
	"Property" : "Volume",
	"Value" : "0.100000",
	"PropertyType" : "IFCVOLUMEMEASURE"
	}, {
	"Property" : "Gross footprint area",
	"Value" : "0.000000",
	"PropertyType" : "IFCAREAMEASURE"
	}
	]
	},
	{
	"Instance" : -1,
	"Ifctype" : "IfcFooting",
	"GlobalId" : "1M2ceu0029mZ4qD3CrCZGp",
	"Name" : "FOOTING",
	"Description" : "1500*1500",
	"FatherGlobalId" : "0WCujW4VT8Mg6cuJAkxH8Z",
	"Children" : [{
	"Ifctype" : "IfcMaterial",
	"GlobalId" : null,
	"Name" : "CONCRETE/Concrete_Undefined",
	"Children" : null,
	"Properties" : []
	}
	],
	"Properties" : [{
	"Property" : "Weight",
	"Value" : "2812.500000",
	"PropertyType" : "IFCMASSMEASURE"
	}, {
	"Property" : "Volume",
	"Value" : "1.100000",
	"PropertyType" : "IFCVOLUMEMEASURE"
	}, {
	"Property" : "Gross footprint area",
	"Value" : "2.300000",
	"PropertyType" : "IFCAREAMEASURE"
	}

	] }] }

PROJEKTIN TIEDOSTON YKSITTÄISEN OSAN METATIETOJEN HAKU

Palauttaa projektin tiedostosta parsitut metatiedot annetulle IFC-objektille.	
URI ja parametrit	/api/projects/{id}/files/{fileId}/versions/{version}/metadatas/{guid} id: Projektin id. fileId: Tiedoston id. version: Tiedoston versionumero. guid: IFC-tiedoston objektin Guid-tunniste.
HTTP-metodi	GET
Http-pyynnön Otsikkotiedot	Authorization: Bearer {Token} Token: Access_token-parametrin arvo, joka saadaan kirjautumisen yhteydessä.
Vastauksen HTTP-statuskoodi	Onnistunut pyyntö: 200 OK

Http-vastaus	{
	"Ifctype" : "IfcColumn",
	"GlobalId" : "1M2ceu0029tZ4qD3CrCZGp",
	"Name" : "COLUMN",
	"Description" : "HEA400",
	"FatherGlobalId" : "0WCujW4VT8Mg6cuJAkxH8Z",
	"Children" : [{
	"Ifctype" : "IfcMaterial",
	"Name" : "STEEL/S235JR",
	"Children" : null,
	"Properties" : []
	}
	],
	"Properties" : [{
	"Property" : "Weight",
	"Value" : "961.100000",
	"PropertyType" : "IFCMASSMEASURE"
	}, {
	"Property" : "Volume",
	"Value" : "0.100000",
	"PropertyType" : "IFCVOLUMEMEASURE"
	}, {
	"Property" : "Gross footprint area",
	"Value" : "0.000000",
	"PropertyType" : "IFCAREAMEASURE"
	}
	]
	}

PROJEKTIN TIEDOSTON GEOMETRIATIETOJEN HAKU

Palauttaa projektin tiedostosta parsitut geometriatiedot.	
URI ja parametrit	/api/projects/{id}/files/{fileId}/versions/{version}/geometrydatas/{guid} id: Projektin id. fileId: Tiedoston id. version: Tiedoston versionumero. guid: IFC-tiedoston objektin Guid-tunniste.
HTTP-metodi	GET
Http-pyynnön Otsikkotiedot	Authorization: Bearer {Token} Token: Access_token-parametrin arvo, joka saadaan kirjautumisen yhteydessä.

Vastauksen HTTP-status-koodi	Onnistunut pyyntö: 200 OK

Http-vastaus	{
	"Id" : 562236f60bc64c1978c07d34,
	"RelatedProjectId" : 562236f60bc64c1978c07d35,
	"RelatedProjectFileId" : 562257700bc65d197803a59a,
	"IfcObjects" : [{
	"Mesh" : {
	"Hidden" : false,
	"Normals" : [{
	"X" : 0,
	"Y" : -1,
	"Z" : 0
	}, {
	"X" : 0,
	"Y" : -1,
	"Z" : 0
	}, {
	"X" : 0,
	"Y" : -1,
	"Z" : 0
	}],
	"Positions" : [{
	"X" : 0.480198026,
	"Y" : -0.303465337,
	"Z" : 0.07945544
	},
	"X" : 0.480198026,
	"Y" : -0.303465337,
	"Z" : -0.174669966
	},
	"X" : 0.474314332,
	"Y" : -0.3028273,
	"Z" : -0.174669966
	}
	],
	"TriangleIndices" : [0, 1, 2, 3, 2],
	"Material" : {
	"R" : 0.89803921568627509,
	"G" : 0,
	"B" : 0.89803921568627509,
	"A" : 1,
	"Name" : "STEEL/S235JR"
	}
	},
	"Ifctype" : "IfcColumn",
	"GlobalId" : "1M2ceu002A8Z4qD3CrCZGp",
	"Name" : "COLUMN",
	"Description" : "HEA400"
	}]
	}

PROJEKTIN TIEDOSTON YKSITTÄISEN OSAN GEOMETRIATIE-TOJEN HAKU

Palauttaa projektin tiedostosta parsitut geometriatiedot annetulle IFC-objektille.	
URI ja para-metrit	/api/projects/{id}/files/{fileId}/versions/{version}/geometry-datas/{guid} id: Projektin id. fileId: Tiedoston id. version: Tiedoston versionumero. guid: IFC-tiedoston objektin Guid-tunniste.
HTTP-metodi	GET

Vastauksen HTTP-status-koodi	Onnistunut pyyntö: 200 OK

Http-vastaus	{
	"Mesh" : {
	"Hidden" : false,
	"Normals" : [{
	"X" : 0,
	"Y" : -1,
	"Z" : 0
	}, {
	"X" : 0,
	"Y" : -1,
	"Z" : 0
	}, {
	"X" : 0,
	"Y" : -1,
	"Z" : 0
	}],
	"Positions" : [{
	"X" : 0.480198026,
	"Y" : -0.303465337,
	"Z" : 0.07945544
	},
	"X" : 0.480198026,
	"Y" : -0.303465337,
	"Z" : -0.174669966
	},
	"X" : 0.474314332,
	"Y" : -0.3028273,
	"Z" : -0.174669966
	}
	],
	"TriangleIndices" : [0, 1, 2, 3, 2],
	"Material" : {
	"R" : 0.89803921568627509,
	"G" : 0,
	"B" : 0.89803921568627509,
	"A" : 1,
	"Name" : "STEEL/S235JR"
	}
	},
	"Ifctype" : "IfcColumn",
	"GlobalId" : "1M2ceu002A8Z4qD3CrCZGp",
	"Name" : "COLUMN",
	"Description" : "HEA400"
	}